NASA CR-134473

(NASA-CR-134473) NASIS DATA BASE MANAGEMENT SYSTEM - IBM 360/370 OS MVT IMPLEMENTATION. 4: PROGRAM DESIGN (Neoterics, Inc., Cleveland, Ohio.) ~~590~~ p HC $31.50 CSCL 09B G3/08

N73-31135

Unclas 13773

577

NASIS DATA BASE MANAGEMENT SYSTEM - IBM 360/370 OS MVT IMPLEMENTATION

IV - PROGRAM DESIGN SPECIFICATIONS

NEOTERICS, INC.

prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

NASA Lewis Research Center
Contract NAS 3-14979

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-134473		

4. Title and Subtitle	5. Report Date
NASIS DATA BASE MANAGEMENT SYSTEM - IBM 360/370 OS MVT IMPLEMENTATION IV - PROGRAM DESIGN SPECIFICATIONS	September 1973
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No. None
	10. Work Unit No.
9. Performing Organization Name and Address Neoterics, Inc. 2800 Euclid Avenue Cleveland, Ohio 44115	11. Contract or Grant No. NAS 3-14979
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	13. Type of Report and Period Covered Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes

Final Report. Project Manager, Charles M. Goldstein, Computer Services Division, NASA Lewis Research Center, Cleveland, Ohio

16. Abstract

The NASIS development workbook contains all the required system documentation. The workbook includes the following seven volumes:

I - Installation Standards (CR-134470)
II - Overviews (CR-134471)
III - Data Set Specifications (CR-134472)
IV - Program Design Specifications (CR-134473)
V - Retrieval Command System Reference Manual (CR-134474)
VI - NASIS Message File (CR-134475)
VII - Data Base Administrator User's Guide (CR-134476)

17. Key Words (Suggested by Author(s))	18. Distribution Statement
	Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	582	$10.75

* For sale by the National Technical Information Service, Springfield, Virginia 22151

TABLE OF CONTENTS

TOPIC A.1 - MONITOR

A. MODULE NAME

Multi-Terminal Tasking Monitor
Program-ID - NMTTSUP
Module-ID - MTTSUP

B. ANALYST

J. H. Herpel
Neoterics, Inc.

C. MODULE FUNCTION

The Monitor is the program which is responsible for the multi-programming and terminal-handling functions for CS NASIS. It also contains support for such things as some user commands which fall most easily to the Monitor to process, the loading/scheduling of programs in "transient" memory, the accumulation of user timing statistics, and so on. Finally, it is the single program which is responsible for the terminal-related communication with the operating system (OS/360).

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

a. Overview

With the exception of some routines which initiate NASIS and take it back down, the Monitor consists of one queue-scanning and scheduling routine to scan the various queues which tell the Monitor what work there is to be done and a bunch of subroutines to actually perform the indicated work. Thus, there are two main flows of control through

the program. The first
is from the queue-scanning/scheduling routine to a subroutine to perform
a particular function (program load, terminal I/O, etc.), back to the
scheduler and by and by to a (NASIS) task for execution. Since the Monitor
can be called by NASIS programs for requests, the other possible flow has
a call to the Monitor at one of the service entry points and a return to
the calling program around the first flow. Finally, under certain circum-
stances, the scheduling routine is not always needed to fulfill an
application request so it may be left out of the flow of control on
occasion.

b. External Specifications

1. Module Name - MTTSUP
2. CSECT name - MTTSUPC
3. Entry Point Names

   a. OSNASIS (To initialize all of NASIS.) Main entry.
   b. MTTREAD (To read text from a user terminal.)
   c. MTTWRITE (To write text to a user terminal.)
   d. MTTPGMIN (To process program interruptions.)
   e. MTTWREAD (To write text to and read text from a user terminal.)
   f. MTTFLUSH (To empty the I/O buffer to a user terminal.)
   g. MTTTSEND (To end the time-slice for a user.)
   h. MTTXTR (To provide application information to a program.)
   i. MTTGETIM (To provide timing information to a program.)
   j. MTTMUST (To enter "must complete" status.)
   k. MTTKA (To enter "KA" mode.)
   l. MTTKB (To enter "KB" mode.)
   m. MTTUSERS (To execute the "users" command.)
   n. MTTNUSER (To execute the "nusers" command.)
   o. MTTHELP (To execute the "help" command.)
   p. MTTMSG (To execute the "msg" command.)

q. MTTPASS (To provide a program with a user's security code.)
r. MTTNLOD (To provide skipping of transient loading.)
s. MTTMLOD (To provide requirement of transient loading.)
t. MTTCALL (To support the loading of a transient program.)
u. MTTITSE (To leave "must complete" mode.)
v. MTTCMND (To sort and process a user command.)

4. External References

a. TSATIN (Application attention processing.)
b. IHEMAIN (For initializing a task.)
c. IHENTRY (For initializing a task.)
d. MASTAB (Pointer to the master set file table.)
e. INTFNDS ("FINDDS" table initialization routine.)
f. TSATIN (Application attention processing.)

c. Sectional Narratives

The following sections describe the workings of the Monitor section by section as the sections appear in the Monitor itself. Please note that these discussions are ordered only by their appearance in the Monitor and not in any manner of program flow.

1. MTTINIT (System Initialization)

This routine is the one called by the operating system to initiate NASIS. it first sets-up the proper linkage and then loads and executes the MTTSTART routine to perform all the once-only initializing functions. After the MTTSTART routine has finished execution and returned to the Monitor, it is unloaded (deleted). Now a SPIE is issued to obtain control upon the occurance of a non-maskable program interruption. (The maskable interrupts are masked out.) After this is done, some alterations are made in a couple of BTAM modules. Specifically, IGG019PF is altered to permit

attention interrupts and IGG019MP is altered to prevent timeouts. At this point all the initialization is complete and the scheduler (MTTFINDQ) is called to begin user processing.

2. INITEND (System termination)

This routine un-does everything that MTTINIT did. It is called by the scheduler after it has determined that it is time to shut down. All this routine does is load, execute and delete the MTTEND routine and then issue a SPIE macro with no operands to relinquish control of program interruptions. After this is done, the routine returns control to the operating system by returning through the linkage.

3. MTTFINDQ (Queue-scanning and scheduling)

This is the routine which looks for work for the Monitor to do and calls the appropriate subroutines to perform the indicated tasks. On entry, this routine saves registers for returning callers and sets an indicator as to whether or not there are saved registers. It then zeroes out the current-user indicators preparatory to switching tasks.

The first scan made is the one for messages to be sent to tasks. If any are found in the Terminal Table (TRQ) and the task is eligible to recieve the message, it is sent by calling the MTTSENDM routine.

After this scan is done, the current time is obtained by the TIME macro and compared with the time specified at shutdown time. If the time returned by TIME is greater than the shutdown time, the INITEND routine is called to terminate NASIS.

The next scan is again through the TRQ for users to be forced off NASIS. If any are found, a quickie subroutine named FDQFORCE is called to send the

force message to the user (MTTWRIT1) and then log off the user (MTTQUIT1).

The next scan through the TRQ looks for user core requests. They are processed by a call to the MTTCORE routine.

Now the scan through the terminal table for completed user terminal I/O operations is started. This scan is made by interrogating the DECB in each TRQ entry for posted status. Each time one is found to have been posted, it is examined by branching to the appropriate code to process each permissible terminal type. Each time a DECB is found not to be posted, it is returned to the ECB list to be waited on if there is I/O in progress for the terminal. If I/O is not in progress, the TRQ is skipped over. The I/O completion examination code starts with a check for a valid terminal type. After this is verified, the code for the type of terminal is called to check for exceptional (error) conditions (i. e. line errors). In the case of a hard line error, the line is hung up and re-opened; in the case of an attention, indication of same is merely added to the return code returned to the caller of the scanning routine. If there were no errors, examination continues by checking the type of operation which just completed. If it was a read initial operation, the MTTTASKI routine is called to log on the new user. If it was a write operation, processing is finished and the FDQRETN routine is branched to to enter the task into the work-to-be-done queue. If the completed operation was a read, the returns are checked to see if the user entered too much text (buffer overflow). If that is the case, a return code is set-up to indicate the condition to the caller of the scanning routine. After the I/O completion stati are processed, the task is placed into the work queue (this is

FDQRETN). This is done by locating the TRQ entry pointer and setting the work flag in it so that the next scan will find it. This scan is finished when the entire list of DECB's has been examined.

The last scan is the one which looks for tasks with processing to be started. These can be tasks restarting after a time-slice end or tasks which have gone through the I/O completion code described above. The TRQ is scanned for users with work to do starting with the task after the last one to have been dispatched. After a task has been found with work to be done, the loading indicators are examined to determine whether or not a transient load is required to get the appropriate program back in memory. If a load is needed, the MTTLOAD subroutine is called to process it. If on return from that routine, the module is loaded, the FDQTSISP routine is branched to to get the task started. Otherwise (or if the task didn't have work to do) the rest of the TRQ is scanned.

If a task was found eligible to start processing by the last scan, all the pointers relavent to the "current task" information are posted and the dispatcher (MTTDISPR) is branched to to actually initiate execution for the task.

If the last scan didn't find anything, it means that there is nothing at all for the Monitor to do. At this point, a timer ECB is added to the end of the list of terminal ECB's (for checking shutdown time) and this ECB list is WAITed on. Upon return from this WAIT, the timer is cancelled and the beginning of the queue scanner is returned to to see which ECB it was that was posted.

Also in the queue-scanner is the routine to post the completion of the timer set by the waiting routine. All it

does is post the ECB for the timer and return to the system so as to notify the wait routine that it terminated.

4. MTTLOAD (Transient module load)

This is the routine which checks and (maybe) loads a transient module. Upon entry it investigates the segment table to see if a SEGLD has been posted complete. If one has, the internal flag is turned off. Now the module to be loaded is looked for in the Module Entry Table (MET). After it is found, the MET entry is examined to see if the requested module is already loaded. If it is, the caller is returned to with the loaded flag set in the TRQ entry. If the module is not loaded, a check is made to see if it has been assigned to a partition. If it has, a check is made to see if the module has been loaded (segment table). If it has, the caller is returned to; if the load is in progress for the module, return is made to tell the scheduler to wait a while. Otherwise, the code to check for a timed-out partition is branched to.

If the requested module is not assigned to a partition, an empty partition is looked for. If one is found, the module is assigned to it and the code to initiate the loading of the module is branched to. If no empty partition is found, one which is not being used is looked for (i. e. the number of users is zero). If one is found, its module entry is marked unloaded and the code to load the requested module is branched to. The next effort to locate a partition is to find one which has timed out. If one is found which has exceeded its time in execution, it is overlayed in the same manner as an unused partition. The last test is to see if the requested module is a "priority" module. If it is, the first partition is overlayed with the requested module in the same manner as

if the partition was unused. If all the test failed, the module is marked not loaded and the scheduler is returned to.

The code to initiate a load into a partition begins by posting the MET pointer in the partition table (PET). Then the partition number is set in the MET and the partition time is set to zero. Now a check is made to see if a SEGLD is already in progress since only one is allowed to be in progress at a time. If there is one already executing, the scheduler is told that it will have to wait for a while. Otherwise, all the parameters to initiate a SEGLD are set-up manually and the SEGLD for the module is initiated via a SVC 37. Now the internal flag indicating a SEGLD in progress is turned on and the scheduler is returned to with the load flag in the TBQ entry turned off to tell it to wait until the module is loaded.

5. MTTTIMER (Sub-task time-slice timer initiation)

All this routine does is start the task's time-slice timer running with the STIMER macro. The interval used is the"next time slice" field in the TCTE. After the STIMER is issued, an internal flag is set indicating that there is a timer active and the caller is returned to.

6. MTTUNTIM (Sub-task time-slice timer cancellation)

This routine is called whenever somebody wants to stop the task's time-slice timer (for instance, a service routine called by the application). On entry it makes sure there is a timer running (if there isn't, the caller is merely returned to) and cancels it with a TTIMER macro call. After the TTIMER completes, the next time slice value is updated to contain the amount of time left in the time-slice and the timing accumulators are updated by

the amount of time actually used from the time slice. Return is to the caller.

6. MTTTSEND (Time-slice end)

This is the routine specified at the exit routine by the STIMER issued by the MTTTIMER routine. Upon being called by the operating system, it "dequeues" the timer interrupt by saving the registers and return address in the PRB and returning to the operating system. After this manouver, the task is placed into the work queue (TRQ) and all the timing accumulators are updated. The user is set-up to get an entire time-slice next time and the queue-scanning routine is exitted to (at MTTFNDQ1).

7. MTTTSE (Forced time-slice end)

This is a routine which the application may call to prematurely end a user's time slice. After performing linkage initialization, it calls the MTTUNTIM routine to turn off the time-slice timer. Now it moves all the caller's registers and a return PSW into the task's TCTE. Finally, the task is marked dispatchable in the TRQ and the queue-scanner is called to look for something to do. (When it re-dispatches the calling task, it will dispatch it to the return address presented to MTTTSE.)

8. MTTTASKI (Sub-task initialization)

MTTTASKI is the routine called by the scheduler when it finds the completion of a read initial operation at a terminal. After checking to see if the system is in the midst of a shutdown (if it is, the request to log on a user is ignored), the routine obtains space for the user's task control table (TCTE) and posts its pointer in the terminal table. The user counter is incremented as soon as this action is completed for consistency with the logoff routine. Now the TCTE is

initialized with as many fields as
the routine knows about. The user at the
terminal is prompted for his
NASIS-id and (optionally) his password by
calls to MTTWRIT1 and MTTREAD1.
(If he can't provide a valid NASIS-id or
password in three tries, he is
logged off by a call to the MTTQUIT routine.)
After these items are
obtained and posted, the remainder of the
information necessary for the
TCTE is filled in, the operator is notified
that another user has joined
the application and the user is notified of
impending shutdowns and
given the news of the day by calls to
MTTWRIT1. Finally, he is marked
present to the application and all his
registers are initialized in
the TCTE along with some of the vital
indicators. Return is made to
the queue-scanner at MTTFNDQ1.

9. MTTQUIT (Sub-task termination)

This is the routine used to terminate a
sub-task. It is called either by one
of the routines to force a user off the
system or by a normal return from
the application task. As this routine may or
may not have to return to its
caller, it sets-up indicators for this when
it is called. It then checks
to see if the task to be quit has a task
control table. If it does not, the
implication is that somebody hung-up a phone
and the line is merely re-
enabled. Otherwise, the operator is notified
as to who is leaving by the
MSG macro. Then the user is removed from the
user table (his logged-on
flag is zeroed out so somebody else can use
the userid later). Now the
logoff message containing the used connect
and CPU times is built and sent
to the user via a call to MTTWRIT1/MTTFLSH1.

At this point, the TCTE space is freed and
the user is taken out of the
terminal table (TRQ). The count of active
users is decremented and the
space used to build the logoff message in is
released. Now the line the

user was using is re-enabled by issuing a read initial on it. After all this is finished, either the caller is returned to or if there is no return to be made, control is returned to the queue scanner at MTTFNDQ1.

10. MTTPOST (Sub-task attention processing)

This is the routine which the queue-scanner calls with it finds a terminal which received an attention-only status. (This means that there was an asynchronous attention which is a special case.) Upon entry, it locates the TCTE for the task and checks myriad and sundry conditions to see if the attention should be ignored (shutdown, non-logged-on user, etc.). if the interrupt is acceptable, space for an attention table is obtained and all the interrupt information is moved into it. it is then chained onto the end of the chain of such tables and the application attention processing routine is called with this table is input. If this processor returns to us, the last table on the chain is un-linked and freed. Now the task is reset by posting the attention interrupt information back into the TCTE and posting the work-to-be-done flag in the TRQ. Exit is to the queue-scanner at MTTFNDQ1.

The other entry to this routine is at MTTPOST1 for those routines which are returned with the attention-in-addition return code set. This entry merely "fakes" the interrupt information in the TCTE and then processes the attention as above.

11. MTTSENDM (Message sending subroutine)

This subroutine is called by whomever decides it is time to send a message to one of the sub-tasks. All it does is locate and describe the message, call MTTWRIT1/MTTFLSH1 to send it, take the MCB (message control block) out of the MCB chain and free the space for the MCB and the message by the

FREEMAIN facility. Return is to the caller.

12. MTTCHEKM (Message checking subroutine)

This subroutine is the one called by several of the service routines to see if there any messages pending for the sub-task involved. If there are, the MTTSENDM routine is called to send the top message on the chain and get rid of the MCB and message space. Return is to the caller.

13. MTTGETIM (Timing obtaining routine)

This service routine is called by the application whenever it wishes to find out how much time the sub-task user has spent connected to the application and actually computing. After it initializes the linkage, it calculates the user's connect time by subtracting the current system time from the time when the user logged on. The user's elapsed CPU time is taken directly from the TCTE (TCTCPUTM). Both these results are placed in the parameter list provided by the caller and he is returned to.

14. MTTXTR (Information extracting routine)

This is the service routine to present to an application program some information about the user using the sub-task. Into a parameter list provided by the caller, this routine places the following information: the NASIS-ID of the user, the password of the user, the task-id of the sub-task, a flag to say whether multi-tasking is active (this flag is always set on), a flag to say whether or not the sub-task is conversational (this flag is always set on). This routine, after filling in the parameter list for the caller, returns to him.

15. MTTNLOD (No-load request routine)

This routine is called by those application programs which wish to indicate to the Monitor that no transient load is

required to re-dispatch the user.
All this routine does is set the "no-load" flag on in the task's TRQ entry after it performs standard initialization. Return is to the caller through standard linkage.

16. MTTMLOD (Reset no-load request routine)

This service routine is the inverse of MTTNLOD. After performing initialization, it turns off the "no-load" flag in the user's TRQ entry, turns on the "load" flag (so that no messages are sent) and then calls the scheduling routine (MTTFINDQ) to cause the module whose name is in the user's TRQ entry to be loaded. After the scheduler returns, the implication is that the module in question has been loaded, and return is made to the caller through the linkage.

17. MTTCALL (Module call routine)

This is the routine used by the application "call" routine (DBCALL) to get a module loaded. Except for the resetting of the "no-load" flag (which DBCALL has set for us), the action of this routine is identical to the action of the MTTMLOD routine.

18. MTTMUST (Must-complete routine)

This routine comprises the "must-complete" function for the application. After this routine has been called by an application program, the sub-task involved will execute until either MTTTSE is called or until the next terminal transaction takes place. All this routine does is initialize the linkage, turn off the user's time-slicing timer and return to the caller.

19. MTTPASS (Password obtaining routine)

This routine is called by the application program which needs to know the user's (new) password (security code). After it initializes itself and the linkage, it uses MTTWRIT1 and MTTREAD1 to

obtain a new security code from the user. ("Blanking" is used here as it is for the password at logon time.) No checking of the entered security code is performed; after it has been read, the caller is returned to through the linkage.

20. MTTPGMIN (Program interrupt handler)

This is the routine which is called as a result of the SPIE issued in MTTINIT to process the occurance of a program interruption. After it sets-up all the necessary registers to run with and saves the interrupt registers and PSW, it returns momentarily to the operating system to cause the interrupt to be "dequeued". After the return from this excercise, a message is built to send to the operator listing out all the interrupt registers and PSW (for de-bugging purposes). Then a message is sent to the user indicating that the system has done him a wrong and he is forced off the application by a call to the MTTQUIT routine. (If there was no user in control when the interrupt occured, this last step is skipped.)

21. MTTWRITE (Sub-task write)

This is the application service routine to write text to the user's terminal. After this routine has initialized the linkage (for external callers), it locates and moves the parameters (text pointer/text length) into registers. (At this point is the entry for internal callers who already have their parameters in the appropriate registers.) Now some internal register/flag initialization is performed and the text length is checked. If the length is zero, a single carriage-return is written to the user. Now all trailing blanks are removed from the text by shortening the text length.

Now the buffer is checked for any room at all and if it is full, it is emptied by writing it out to the terminal with the MTTFLSH1 routine.

At this point, the initialization for teletypes and for writing after reading is performed. For writing after reading, a number of idles corresponding to the distance across the paper the carriage was when the user hit carriage-return is written out. For a teletype, if the last line was not carriage hanged, a line-feed and an idle are written out.

Now the text is processed character by character. One character is picked-up, tested and put into the I/O buffer in the task's TCTE. For certain characters there is additional processing performed. These are:
Line-feeds, after which idles must be added; Backspaces, which must be accounted for in the distance across the paper the carriage is; carriage returns, after which must be added idles; tabs, after which must also be added idles. To speed up the testing process, there is an additional branch high after the test for line-feeds (which is the first test) which will kick out most of the text. For teletypes, a line-feed may be added after the carriage-returns found in the text.

After the text has all been moved, end processing begins. If the text ended with a carriage hang character (":"), this processing is skipped. Otherwise, a carriage-return and idles are inserted after the text. Also, in the case of a buffer which has just filled up with the last character, the buffer is emptied to the terminal with a call to MTTFLSH1.

After this has been done, all that remains is to remember whether the routine was called internally or externally and return appropriately.

22. MTTCHARS (Character stuffing subroutine)

This subroutine is used by MTTWRITE whenever it wants to add s discreet number of characters to the I/O buffer. As it moves these characters, it

checks the condition of the buffer and flushes it to the terminal with MTTFLSH1 if necessary. Return is to the caller (in MTTWRITE) through a linkage register.

23. MTTREAD (Sub-task read)

This is the service routine to read text from a user's terminal. After initializing the parameter list pointers (one for the target area and one for the input length area) for external callers, the code for both external and internal callers joins as with MTTWRITE.

If there is any text in the user's I/O buffer, it is written with a call to MTTFLSH1 specifying that the queue routine to be used is the one which reads the terminal after writing the text to it. If there is no user output text in the buffer, the MTTRDQ routine is called to just read the terminal.

After the text has been read, one way or the other, the length is checked to make sure the user didn't overflow the buffer. The text is now translated from line code to EBCDIC and, optionally, all lower-case alphabetics are translated to upper-case. If there is a carriage-return at the end of the text it is removed at this point.

The next thing checked for is a cancelled line. If the last character in the user's text is the line-kill character, the routine re-calls itself to begin reading the terminal over again.

Now backspaces and trailing blanks are processed and stripped off, respectively. Also, a check is made to see if the last character in the text is the continuation character, in which case the flag in the return code for that event is turned on. (The application programs are left to process continuation however they wish to.)

If it was the operator's terminal which was

read, his input is scanned to see if it is one of the Monitor commands. If it was, the appropriate processor is located and called and the routine to re-call the MTTREAD routine is called.

Finally, the length actually read in is checked against the target length sent by the caller and the text is moved to the caller's target area. A flag indicating text truncation is turned on if the user entered more text than the caller allowed for.

Now determination is made to see whether the routine was called internally or externally and appropriate return if effected.

24. MTTWREAD (Sub-task write/read)

This routine is the application service routine which is used to write text (a prompting string) to a user's terminal and then read in the response to the text. It is not internally callable.

After initialization of the linkage is done, this routine merely calls the MTTWRIT1 and MTTREAD1 routines with the parameters supplied by the caller to perform the necessary functions. Return is to the caller through the standard linkage.

25. MTTFLUSH (Sub-task flush)

This is the service routine which the application or the Monitor itself can call to cause the task I/O buffer to be emptied to the user's terminal. There is initialization code for external callers to get the linkages set-up properly, for internal callers, the registers are merely saved. After this is done, the routine checks to see if there is any text in the buffer. If there is none, the caller is merely returned to. If there is, and the terminal is a 1050, an end-of-block (EOB) is added to the end of the text. Now the text is ready to be

written out.

For internal callers, the address of the desired queue-writing routine is already specified. For external callers, MTTWRQ is assumed. The queue-writing routine is now called to write out the text and the return code is checked. If the return code is zero (successfull initiation), the queue-scanning routine (MTTFINDQ) is called to await the completion of the I/O operation. After the queue-scanner returns, the caller of flush is returned to (depending on how the routine was called).

26. MTTRDQ, MTTWRQ, MTTWRQAR (Queue I/O routines)

These are the routines which actually perform writing and reading I/O on the sub-task terminal. They are called only from within the Monitor itself.

MTTRDQ is the routine which is used to only read text from a sub-task terminal. After saving all the registers and locating the DECB in the task's TRQ entry, this routine executes a BTAM read (TV) on the DECB. It then turns off the flag allowing WRITE type "continue"s on the terminal and branches to join the common return processing code.

MTTWRQ is the routine to only write text to a terminal. After it saves the registers and sets-up the DECB pointer in the TRQ entry, it determines whether a "conversational" or "continuation" write is called for and performs the appropriate action with a BTAM WRITE macro. In addition, if it is the operator's terminal being written to, the completion of the write is awaited with the WAIT macro. The continue-write-permitted flag is turned on in all cases and control transfers to the common return processing section.

MTTWRQAR is the routine to read text from a

user terminal after writing text to it. This is accomplished by first issuing a BTAM WRITE (as in MTTWRQ) and then a BTAM type TV READ. After this is initiated, the continue-write-permitted flag is turned off and the return processing section is fallen through to.

After whatever type of I/O operation has been initiated, the return code from the WRITE or READ is tested. If it is non-zero, an I/O error return is sent back to the caller. If it is zero, the terminal is marked busy in the TCTE and the caller is returned to.

27. MTTMOVE (Text moving subroutine)

This internal subroutine merely moves text from here to there in an efficient manner. The inputs are to address, from address and length. The method used are successive MVC's of length 256 until there are fewer than 256 bytes to be moved and then an executed MVC is issued to move the rest. Return is to the caller through a linkage register.

28. MTTTRAN (Text translating subroutine)

This internal subroutine is used to translate text. The inputs are text address, table address and length. The method used is similar to the one for MTTMOVE except that TRs are used instead of MVCs.

29. MTTCMND (User command routine)

This routine is called by the application to execute a user command. It takes as input parameters describing the command processor address, the data text pointers and lengths and calls the appropriate command processor internally. After the command is finished, return is made to the caller through the standard linkage.

30. MTTKA (KA command routine)

This processor is the routine to execute the

KA command. All it does is
turn on the KA flag in the task's TCTE and
return to the caller

31. MTTKB (KB command routine)

This routine processes the user KB command in the same manner as the MTTKA routine except that the flag is turned off instead of on.

32. MTTMSG (MSG command routine)

This routine is the processor for the user/operator MSG command. It first locates and verifies the receiving userid parameter with the MTTGTUSR subroutine. It then verifies that there is a message text parameter by checking its length. If either of these tests fail, an error message is send either with MTTWRIT1 (for users) or the MSG macro (for the operator).

Now enough space for the message itself plus the message control blcck (MCB) is obtain via the FREEMAIN macro. The MCB is filled in and chained to the end of the MCB chain, the time-field in the message header is filled in and the flag in the TRQ entry for the user indicating that there is at least one message in the queue is turned on. Return is to the caller through the linkage register.

33. MTTBCST (BCST command routine)

This routine is the one which processes the operator BCSTccmmand. After parsing the input string to locate the message text, it obtains and builds MCBs for each active user in the same manner MTTMSG builds them for single users. After all the active users have the MCB added to their chains, the caller is returned to through the linkage register.

34. MTTSTOP (SHUTDOWN command routine)

This routine is the processor for the operator SHUTDOWN command. This command

is the one used to terminate the application after an optional time period.

After locating the time parameter (and defaulting it if it is not present), the routine converts the time parameter to a fixed number and stores it in the field which is interrogated by the queue-scanning routine. (There is a special code which is used to terminate the application immediately which is also checked for. If it is found, termination is effected by an immediate transfer to the INITEND routine.)

Now the time-of-day of the shutdown is calculated and a message for the users warning them of the shutdown is composed. After the current time is obtained, the shutdown time field is updated to correctness by adding its contents to the current time. After turning on the shutdown-issued indicator, this routine sends the shutdown message to the operator and then exits to the MTTBCST routine to cause the warning to be sent to all the users.

35. MTTHELP (HELP command routine)

This routine processes the user HELP command which is merely a MSG command with the operator assumed to be the receiving userid. This routine merely points to a field containing the operators userid and the text sent by the user and exits to the MTTMSG routine to actually send the message.

36. MTTUSERS (USERS command routine)

This routine is the processor for the user/operator USERS command which is used to list out all the active application users. Upon entry, this routine obtains a workarea with the GETMAIN macro and sets-up to start locking for users in the terminal table (TRQ).

Before the search is started, a header message is moved to the workarea, filled in with the time and then sent to the user with MTTWRIT1 or to the

operator with the MSG macro. Now the terminal table is scanned and for each active user located, his userid, the symbolic device address of his terminal and the user's taskid is sent to either the user or the operator. After all the users have been listed, the caller of this routine is returned to through the linkage register.

37. MTTFORCE (FORCE command routine)

The FORCE command is used by the operator to gracefully get rid of an application user and this is the routine that processes the command. All it does if verify that the userid specified by the operator is a valid one, locate the TRQ entry for this user and turn on the force-issued flag in that table. (The queue-scanning routine will do the actual dirty work.) Return is to the caller through the linkage register.

38. MTTNUSER (NUSERS command routine)

This is the processor for the user/operator NUSERS command which is used to present the number of active application users. All it does is fill in a skeleton message with the current time and number of users (from MTTUSER#) and then send the result with either MTTWRIT1 or the MSG macro depending on whether it is going to a user or to the operator. Return is to the caller through the linkage register.

39. MTTGTUSR (User locating subroutine)

This subroutine is called by various command processors to locate the TRQ entry for a given userid. It compares the userids in all the TRQ entries to the one provided by the caller and returns the correct one if it can be found in the terminal table. If the appropriate user can not be located, a non-zero condition code is returned to the caller to so indicate.

40. MTTBUSR (User error subroutine)

This subroutine is a convenience item used by the command routines when they come across an invalid userid. All this routine does is send an error message to either the user (MTTWRIT1) or the operator (MSG macro) and then return to the original caller of the command routine.

41. MTTBMSG (Message error subroutine)

This subroutine does the same thing as the MTTBUSR subroutine except that the error condition located for this routine was missing message text.

42. MTTPRMPT (Operator communication)

This is the routine which is called as a result of the issuance of an MSG macro. It first sets-up and fills in a time-stamp and sends it to the operator via the MTTWRIT1 routine. Now the message proper is sent to the operator the same way. (The time-stamp is sent with a carriage-hang so that the message comes out all on one line.) Before returning to the caller, this routine turns off the time-slicing timer just in case.

43. MTTCORR (Core obtaining subroutine)

This is the subroutine which is called by the queue-scanner/scheduler when it finds an outstanding user core request. All it does is load the number of bytes requested from the TCTE for the task and issue a conditional GETMAIN macro for that number of bytes. If the GETMAIN is successful, the core request flag in the TRQ entry is turned off before returning, otherwise, the caller is merely returned to.

44. MTTDISPR (Sub-task dispatch)

This is the routine which is called when it is necessary to re-start execution of one of the sub-tasks. Upon entry, it checks for a dispatch for the first

time and exits to the first-time code if this is the case. The other test made here is for an internal dispatch. If that is the case, the registers are merely restored from the task control table (TCTE) and the returning register is branched through.

Normal dispatch to a sub-task is accomplished by restarting the task timer (MTTTIMER routine), loading the floating registers, loading the general registers and finally entering supervisor state long enough to load the resume PSW which has been moved to location 240 (decimal) for addressibility.

First-time dispatching is handled by first turning off that indicator and then starting the task's time-slice timer (by calling MTTTIMER). After the floating registers are initialized, linkage is set-up to call the application the first time by pointing IHEMAIN at the application entry point (DBMTT) and calling it.

TOPIC A.2 - INITIAL ENTRY ROUTINE

A. MODULE NAME

Initial Entry Routine, Retrieval Only
Program-ID - NDBMTT
Module-ID - DBMTT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The function of this module is to perform the necessary allocations of the external data items used by the retrieval system. It also issues the initial prompt, which is used to determine which NASIS sub-system the user wishes to invoke, and then calls the proper module for that sub-system.

D. DATA REQUIREMENTS

1. I/O Block Diagrams

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

The program makes use of the following tables:

a. USERTAB

b. VERBTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Initialize

This routine initializes the interrupt (ATTN and END) processing routines and the PL/I error handler. It allocates and initializes the user data table. The program also allocates and initializes the verb table (including user specified commands) which it uses in the prompt routine.

b. Define

This routine performs all of the file control block allocations and initializations necessary for the proper operation of the rest of the NASIS system.

c. Prompter

This routine sets a temporary END condition handler which results in a new prompt on an END condition. It prompts the user for a command and searches the verb table for a matching entry. If no match is found a diagnostic message is written to the user and the prompt re-issued.

The verb table entry is analyzed and if an immediate command has been entered, the program branches to the routine which processes that command. Otherwise, the program optionally establishes a new strategy and then calls the entry point of the

processer for the command entered. When control is returned to DBMTT, the user is prompted for the dispcsition of the current strategy and it is either renewed or erased.

When the command entered has been completely processed, control is passed back to the prompting routine. The entry of an END command causes the program to be terminated.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in the IBM PL/I (F) language.

2. Suggestions and Techniques

Not Applicable

TOPIC A.3 - MT/T MONITOR INITIALIZATION ROUTINE

A. MODULE NAME

Monitor Initialization Program
Program-ID - NMTTSTRT
Module-ID - MTTSTRT

B. ANALYST

J. H. Herpel
Neoterics, Inc.

C. MODULE FUNCTION

The Monitor Initialization Routine is loaded by the MT/T Monitor to perform all the once-only initialization function required by the Monitor. Since this routine is loaded, executed and then unloaded, the space taken up by it is not left idle after it is finished executing.

D. DATA REQUIREMENTS

1. Parameter list from the EXEC statement in the JCL stream.

2. DD statement for the master set file information table in the JCL stream.

3. DD statement for the module table in the JCL stream.

4. DD statement for the ESD table in the JCL stream.

5. DD statement for the user table in the JCL stream.

6. DD statements for the terminal line groups in the JCL stream.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

a. Overview

This program performs all the once-only

initialization required by the Monitor. These include the obtaining and setting-up of the user table, module table, partition table, line groups and so on. There is direct communication with the Monitor by way of an entry point into the Monitor's CSECT where the communication variables are. This routine is also responsible for initializing all the terminal lines and initiating the operator's sub-task.

b. Detailed Description

The following section is a discussion of the program in the order it appears in the listing.

After MTTSTRT has initialized the linkage conventions, the first thing it does is inform the (OS) operator where the Monitor's CSECT begins by printing its address on the console typewriter with a WTO macro. It then parses and isolates the parameters supplied on the EXEC card in the JCL stream.

One of the parameters is the maximum allowed number of users. This parameter is used for allocating space for and initializing the terminal table (TRQ). Space for it is obtained via GETMAIN and then the following fields are initialized in each entry: TCTE pointer, module request field, flags, module table entry pointer, relative terminal number, skeleton DECB and a fake symbolic device address.

From the TIOT (Task I/O Table) the descriptions of the line groups are located (they get there from the DD statements in the JCL stream). These are used to fill in terminal information in the TRQ terminal type and DECB fields.

Next, the routine locates the dcb for the Module Table and opens it. It then reads the header record from the table and from it calculates the room necessary to hold the information in the

dataset. After this space
is obtained via GETMAIN, the partition table is
initialized (with the
number of partitions contained on the first record
in the module table).
Also from that record, space for the module table
is allocated with
GETMAIN. Now the module table is built one entry
from each of the
remaining records in the mcdule table dataset.

Now the segment table dataset is located and
opened. The records in it
are scanned until the one describing the CSECT for
the Monitor is found
and its offset is saved in the segment table.

Next the DCB for the user table dataset is opened
and the records in the
dataset are all read in and counted. The number
of records is used to
determine how much space is required for the user
table and that much
space is obtain via GETMAIN. Now the dataset is
read again and entries
in the user table are created from the records in
the dataset, one for one.

After this is done, the set file master table is
opened and read. Space
for the set buffers is allocated. Now space for
the segment table itself
is obtained and the table is initialized from
information in the set file
master table.

At this point, the space for the task control
table for the operator's task
is obtained via GETMAIN. After the space is
obtained, the TCTE is filled in
with arbitrary operator information and the first
terminal table is posted
as being the operator. (This will be the first
phone line called in on.)
After the TRQ entry is posted with the TCTE
pointer, the duplicated
information is transferred from the TRQ entry to
the TCT.

Now the program points to the DCB list it
constructed for all the terminal
lines and opens each of them with the BTAM OPEN
macro. If any DCB

fails to open in this manner, the program terminates with an error.

After all the line group DCBs are opened, the program locates the DCB for the first phone line and enables it via a BTAM READ initial (TI). It now waits until somebody rings its bell. After a connection is made to this line, its ECB is cleared and some flags are marked special to the operator task (which never calls the application but remains dormant to receive messages and make requests.)

As soon as the operator is connected, all the remaining terminal phone lines are enable with READ initials and the Monitor is returned to.

TOPIC A.4 - MT/T SVC

A. MODULE NAME

Program-ID: NIGC253
Module-ID: IGC253

B. ANALYST

John H. Herpel,
Neoterics, Inc.

C. MODULE FUNCTION

This module is a type 1 SVC issued by the MTTSUP monitor to get into protect key zero-supervisor state or back to problem program state. When the SVC is issued, register one is set to zero for supervisor state or one for original program state. Register zero must contain 'NASI' for this SVC to proceed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   Not Applicable

2. Input Data Sets

   Not Applicable

3. Output Data Sets

   Not Applicable

4. Reference Tables

   Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 1.

2. Narrative

   See MODULE FUNCTION (C).

F. CODING SPECIFICATIONS

1. Source Language

This module is written in IBM OS/360 Assembler language.

2. Suggestions and Techniques

Not Applicable

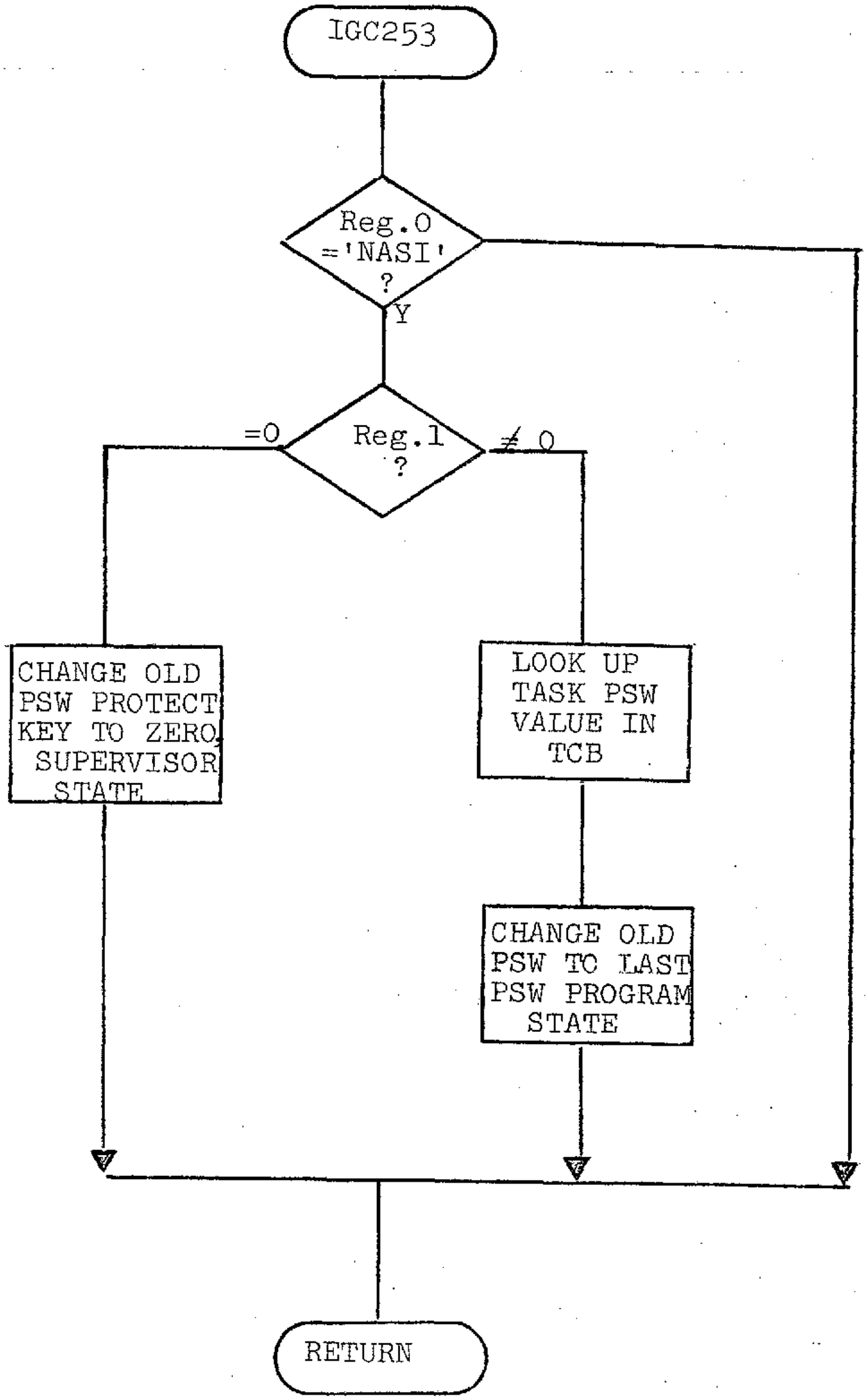


Figure 1. Top Level Flowchart

TOPIC A.5 - INITIAL ENTRY ROUTINE

A. MODULE NAME

Initial Entry Routine, Descriptor Editor Only
Program-ID - NDBMTTE
Module-ID - DEMTTE

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The function of this module is to perform the necessary allocations of the external data items used by the descriptor editor system. It also issues the initial prompt, which is used to determine which NASIS sub-system the user wishes to invoke, and then calls the proper module for that sub-system.

D. DATA REQUIREMENTS

1. I/O Block Diagrams

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

The program makes use of the following tables:

a. USERTAB

b. VERBTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Initialize

This routine initializes the interrupt (ATTN and END) processing routines and the PL/I error handler. It allocates and initializes the user data table. The program also allocates and initializes the verb table (including user specified commands) which it uses in the prompt routine.

b. Define

This routine performs all of the file control block allocations and initializations necessary for the proper operation of the rest of the NASIS system.

c. Prompter

This routine sets a temporary END condition handler which results in a new prompt on an END condition. It prompts the user for a command and searches the verb table for a matching entry. If no match is found a diagnostic message is written to the user and the prompt re-issued.

The verb table entry is analyzed and if an immediate command has been entered, the program branches to the routine which processes that command. Otherwise, the program optionally establishes a new strategy and then calls the entry point of the

processer for the command entered. When control is returned to DBMTTE, the user is prompted for the disposition of the current strategy and it is either renewed or erased.

When the command entered has been completely processed, control is passed back to the prompting routine. The entry of an END command causes the program to be terminated.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in the IBM PL/I (F) language.

2. Suggestions and Techniques

Not Applicable

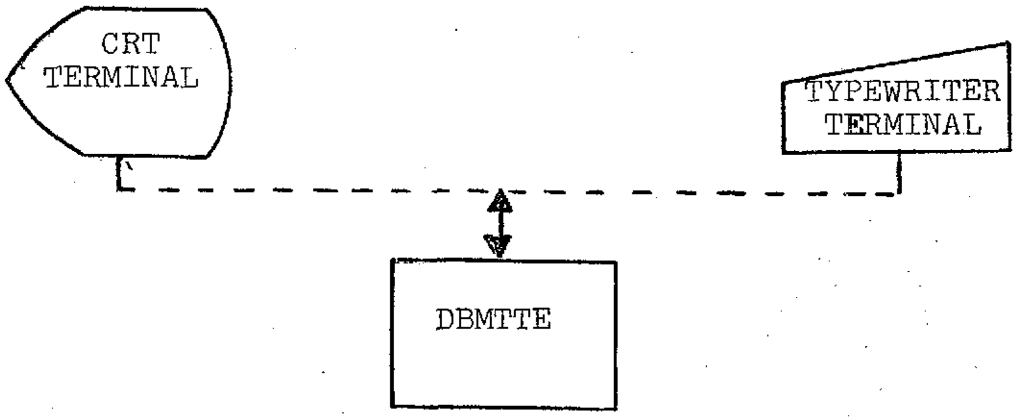


Figure 1. I/O Block diagram

c. Formatted Print-outs

Not Applicable

4. Reference Tables

The program makes use of the following tables:

a. USERTAB

b. VERBTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Initialize

This routine initializes the interrupt (ATTN and END) processing routines and the PL/I error handler. It allocates and initializes the user data table. The program also allocates and initializes the verb table (including user specified commands) which it uses in the prompt routine.

b. Define

This routine performs all of the file control block allocations and initializations necessary for the proper operation of the rest of the NASIS system.

c. Prompter

This routine sets a temporary END condition handler which results in a new prompt on an END condition. It prompts the user for a command and searches the verb table for a matching entry. If no match is found a diagnostic message is written to the user and the prompt re-issued.

The verb table entry is analyzed and if an immediate command has been entered, the program branches to the routine which processes that command. Otherwise, the program optionally establishes a new strategy and then calls the entry point of the

processer for the command entered. When control is returned to DBMTTP, the user is prompted for the disposition of the current strategy and it is either renewed or erased.

When the command entered has been completely processed, control is passed back to the prompting routine. The entry of an END command causes the program to be terminated.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in the IBM PL/I (F) language.

2. Suggestions and Techniques

Not Applicable

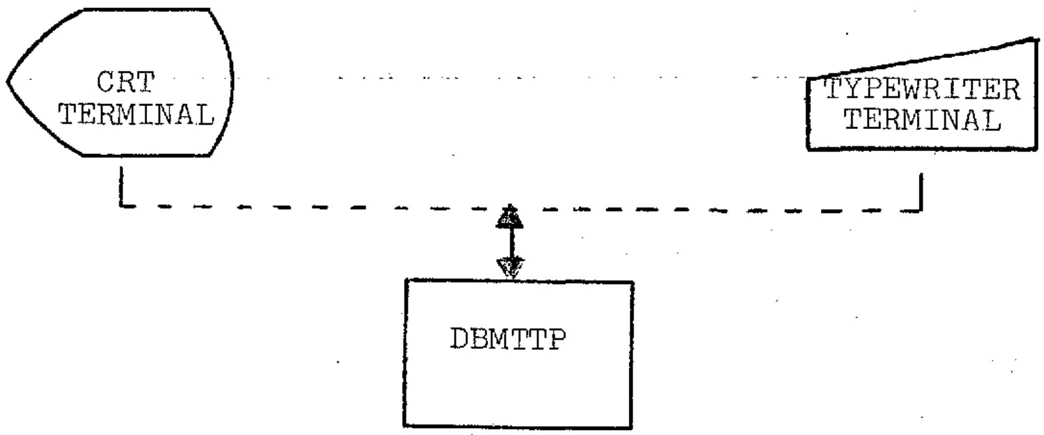


Figure 1. I/O Block diagram

TOPIC B.1 - EXECUTIVE PRE-PROCESSOR

A. MODULE NAME

Data Base Executive - Preprocessor
Program-ID - DB
Module-ID - DB

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

DB analyzes Data Base PL/I language extension (DBPL/I) statements and generates, in their place, in a source program, PL/I statements for communication with the Data Base Executive (DBPAC OR DBLIST). Diagnostic comments shall be generated for errors that can be detected by DB during precompilation.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Job control parameters for operation under OS are those required for PL/I precompilation and immediate compilation. Refer to the appropriate IBM PL/I Programmer's Guide. The PL/I compiler parameters-MACRO, SOURCE2, and COMP (among others) are specified to indicate that precompiling, precompiler input listing and compiling are desired.

b. Punched Card Input Files

1. DB Text

The DB text deck is all text for insertion into the source program as a result of a % INCLUDE DB; statement in the source program. This text is composed of the source statements of the DB preprocessor function procedure, itself, and any PL/I statements to be

unconditionally inserted at the % INCLUDE DB; pcint in the source program. The DB text is coded as specified in this report, formatted according to PL/I source language standards and catalogued once in a data set for compile-time use by all programs using DB.

2. Source Deck

The SOURCE Deck is any PL/I source program using DB for its DBPL/I statements. It is prepared according to the DBPL/I User's Manual (DWB Section V, Topic B.2) to access a self-describing data bâse and formatted according to PL/I source language standards.

c. Input Files

The DB text is catalogued as a member, named DB, of a partitioned direct access data set for retrieval by the IBM PL/I precompiler. The data set is accessed via ddname LISRMAC.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The object module consists of the relocatable machine instructions and constants generated by the PL/I compiler for the scurce program. It is stored as a member of a program library (partitioned data set) for subsequent linking by the OS system linkage editor.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-outs

1. Precompiler Listings

Two precompiler listings are produced: a source listing before precompilation, and any precompiler diagnostics (these

diagnostics are any errors in the use of preprocessor PL/I, not DBPL/I). The appropriate IBM PL/I Programmer's Guide explains the listing formats.

2. Compiler Listings

The compiler listings produced include an intermediate source listing (between precompiling and compiling) and any compiler diagnostics. Any errors in the use of DBPL/I generate diagnostic PL/I comments in the intermediate source listing. Serious DBPL/I errors may result in compiler diagnostics, particularly for undeclared qualified names when DB has suppressed automatic generation of a declare statement. The appropriate IBM/I Programmer's Guide explains the listing formats.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

MFCB - Mainline file control block.
See Section III, Topic B.4, of the DWB.
DBPL/I - Diagnostic comments.
See Section III, Topic B.1, of the DWB.
DBPL/I - DBPAC Interface.
See Section III, Topic B.2, of the DWB.
DBPL/I - DBLIST Interface.
See Section III, Topic B.10.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Top Level

The mainline PL/I source program is required, according to the DBPL/I User's Manual (DWB Section V, Topic B.2), to have a % INCLUDE DB; statement once in the program before all DB preprocessor function references. This statement directs the PL/I precompiler to take text from member DB of the partitioned

data set accessed via ddname LISRMAC and incorporate it into the source program. (See the I/O Block Diagram in Figure 1.).

The DB text includes the following statement:

    ON FINISH GO TO FINISH;

for "automatic" data base file closing. DBPL/I requires that the PL/I FINISH ON-condition be reserved for this purpose.

The DB text declares and activates the DB preprocessor name by the following statement:

    % DECLARE DB ENTRY(CHARACTER) RETURNS(CHARACTER);

The DB text following the end of the DB preprocessor function procedure itself, invokes DB once as follows:

    DB(INITIALIZE)

This statement is a special function reference to be recognized by DB as the first reference (directing DB to initialize itself).

The remainder of this narrative specifies the DB preprocessor function procedure itself, which is depicted in the Top Level Flowchart in Figure 2.

DB receives one argument from a preprocessor function reference: a varying length character string consisting, in general, of labels, comments, valid DBPL/I statements and, possibly, invalid text. DB's objective is to analyze the argument and generate a varying length character string, called the "generated text", consisting of valid PL/I labels, comments and PL/I statements for communication with the Data Base Executive.

If the special argument, 'INITIALIZE', is received, (i.e., the first reference to DB), the Initialize DB routine is performed and a comment, such as:

    /* DB001 INITIALIZATION COMPLETE. */

is returned for insertion into the source program and DB is terminated. Otherwise, the Argument Initialization routine is performed.

Following the Argument Initialization routine is the point where, in general, DB is logically between DBPL/I statements in its processing of the argument. The Find Subargument routine is performed there. If it finds the right parenthesis at the end of the argument, the generated text is returned for insertion into the source program, and DB is terminated. If Find Subargument finds an inter-statement comment, a statement label, or a null statement (simply a semicolon), then the subargument is concatenated to the right end of the generated text (i.e., "passed through" to the intermediate source text), and preprocessor control is transferred back to the inter-statement point. Otherwise, the Process Statement routine is performed, and preprocessor control is transferred back to the inter-statement point.

b. Diagnostic Comment Generation

Wherever this narrative specifies the generation of a diagnostic comment, the following specifications apply. A diagnostic comment is concatenated to the right end of the generated text for insertion into the intermediate source program. If the diagnostic is for an error, the precompiler count of diagnostics is incremented. If more than four errors are detected in one DB reference further processing of that reference is stopped to prevent the possibility of unpaired quotes, parentheses or comment delimiters looping the preprocessor. A diagnostic has the following general format:

```
/* DBnnn diagnostic-message. */
```

The "DB" preceding the message number indicates that the comment was generated by the DB preprocessor. The three-digit message number guides the user to a more detailed explanation of the message which is documented in the DWB Section III, Topic B.1. The diagnostic message is concise, and does

not contain abbreviations.

c. Initialize DB

Precompiler variables for file attributes, file usages and diagnostic counts are appropriately initialized. These variables are subsequently set or incremented as DBPL/I statements are processed and are examined when the finish statement is processed. A precompiler indicator is set to indicate that the FINISH statement has not yet been processed.

d. Argument Initialization

The argument is examined to find the left parenthesis at its beginning. If any other non-blank character is found, a diagnostic comment is generated and DB is terminated. A precompiler variable pointing to the "current argument character" is initialized to point to the character following the beginning left parenthesis. The generated text is initialized as one blank character.

e. Find Subargument

A subargument, as used in this specification, is a substring of the argument that is one of the following classes of syntactic units:

1. The right parenthesis at the end of the argument.

2. A label, including its colon.

3. An inter-statement PL/I comment.

4. A Null statement consisting only of a semicolon.

5. A DBPL/I statement terminated by a semicolon.

6. A syntax error; i.e., none of the above.

A class (5) subargument may contain paired parenthesis (possibly nested) or string constants enclosed in string quotes. A class (6) subargument will be terminated by a semicolon if one is found but will never

include the right parenthesis at the end of the argument.

The Find Subargument routine is used at the inter-statement point in the Top Level Flowchart. The argument is examined beginning at the current argument character and ignoring leading blanks to find the next subargument. A precompiler variable pointing to the beginning character of the subargument, and another indicating its length is characters, is set. The current argument pointer is advanced to point to the character following the subargument.

f. Process Statement

This routine is the kernal of the DB preprocessor; it analyzes a single DBPL/I statement body (i.e., apart from any statement labels), generates suitable PL/I statements for communication with the DBPAC executive and returns preprocessor control to the inter-statement point. The PL/I statements and comments that are generated are concatenated to the right end of the generated text string for subsequent insertion into the intermediate source program.

A diagnostic comment containing the subargument and any intra-statement comments is generated for documentation and for reference in case of other diagnostics. If the FINISH statement has already been processed or if the subargument has a syntax error, an appropriate diagnostic comment is generated, and control is returned to the inter-statement point.

The Find Keyword routine is performed. If it does not find a keyword that identifies a DBPL/I statement, then a diagnostic comment is generated and control is returned to the inter-statement point. If the keyword identifies a SET, FINISH, FREE or ON statement, control is transferred to the relevant specific statement routine. Otherwise, the Find File Clause routine is performed. If the second clause is not a FILE clause, a diagnostic comment is generated, and control is returned to the inter-statement point. The Find File routine

is performed, and control is transferred to the relevant specific statement routine.

1. Find Keyword Routine

A clause, as used in this specification, is a substring of the subargument that is one of the following classes of syntactic units:

-the semicolon at the end of the subargument,

-a comma separating DBPL/I substatements;

e.g., in a multiple OPEN,

-a keyword with an associated parenthesized argument,

-a keyword without a parenthesized argument.

A keyword-with-argument clause may contain paired parenthesis (possibly nested), or string constants enclosed in string quotes.

The Find Keyword routine is used to find the keyword that will identify a statement to branch to the specific statement routines.

2. Find File Routine

The Find File subroutine extracts the file name from a given FILE clause. If the file-name is not a valid PL/I external name, a diagnostic message is generated, and the statement abandoned by control being transferred to the inter-statement point. Otherwise, the precompiler's file table is searched to determine if the file-name has been used previously in the program. If it has not, a new entry is appended to the file table. In either case, a precompiler variable is set to indicate the current file, and control is returned to the point from which Find File was invoked.

3. Specific Statement Routines

Each specific statement routine examines the statement from left to right until the semicolon clause is found. (The CLOSE and OPEN statement routines recognize a comma clause as separating substatements and loop accordingly). The keywords are verified for correct spelling and order. The FREE LIST routine for specific lists recognizes a comma separating list-pointers and loops accordingly. Routines that process a statement having a FIELD clause recognize a comma separating field-name expressions, find the corresponding element in the FROM or INTO clause and loop accordingly. If any error is detected, a diagnostic comment is generated, and the statement abandoned by control being transferred to the inter-statement point.

For those statements having a FILE clause, the precompiler's file table is posted to record the file usage (for analysis in the FINISH routine).

Following successful analysis, each specific statement routine generates PL/I statements for communication with the DBPAC or DBLIST executive and then loops back either to process another FIELD or FREE LIST element, to process another OPEN or CLOSE substatement, or to the inter-statement point, as the case may be. Special processing for the ON and FINISH statements is specified after the general specifications for all other specific statement routines.

For those statements having an entry in the DBPL/I - DBPAC Interface table (Section III, Topic B.2, of the DWB), an assignment statement is generated in the following format:

filename.OPERATION = 'operation'B;

For example, when processing the following argument:

LOCATE FILE(SAMPF) KEYFROM(REC#) ;

The following assignment is generated:

```
SAMPF.OPERATION = '11010000'B;
```

For statements having a FIELD clause, the operation assignment need only be generated once for the statement, even it it contains multiple field names.

For an OPEN statement having a TITLE clause the following assignment is generated:

```
filename.ONFILE = title-expression;
```

If it has no TITLE clause the following is generated:

```
filename.CNFILE = 'filename';
```

For an OPEN statement having an "access" option and/or a "function" option, a bit-string value is assigned to filename.ATTRIBUTES according to the definition of a Mainline File Control Block (described in Section III, Topic B.4 of the DWB); otherwise, the following assignment is generated for an OPEN:

```
filename.FUNCTICN = '10'B;
```

For each field-name in a FIELD clause, an assignment statement is generated as follows:

```
filename.CNFIELD = fieldname;
```

Where the field-name may be an expression, for example, when processing the following argument:

```
GET    FILE(EXAMP)       FIELD('DATEPUB')
INTO(DP) ;
```

The following assignment is generated:

```
EXAMP.ONFIELD = 'DATEPUB';
```

For those statements having an entry in the DBPL/I - DBPAC Interface table, a CALL statement is generated in one of the following formats, depending on

whether the "Arg1" and "Arg2" columns of the table have entries:

```
CALL entrypoint  (arg1);

CALL entrypoint  (arg1, arg2);

CALL entrypoint  (arg1, arg2, arg3);
```

For example, when processing this statement:

```
LOCATE FILE(DAMPF)  KEYFROM(REC#);
```

This CALL is generated:

```
CALL DBPACFV  (SAMPF, REC#);
```

For those statements having an entry in the DBPL/I-DBLIST Interface Table (Section III, Topic B.10), a CALL statement is generated according to the table.

The ON statement routine examines the second clause. If an ERRORFILE clause is found, the Find File subroutine is performed. The statements shown below at the right are generated for the ON statement shown at the left.

```
ON ERRORFILE(f)  GO TO label;

     f.ERROR.ROUTINE = label;

     f.SYSTEM = '0'B;

ON ERRORFILE(f) SYSTEM;

     f.SYSTEM = '1'B;

ON LISTERROR GO TO label;

     LISTERR.ERROR.ROUTINE = label;

     LISTERR.SYSTEM = '0'B;

ON LISTERROR SYSTEM;

     LISTERR.SYSTEM  = '1'B;
```

The FINISH statement routine sets a precompiler indicator to indicate that a

FINISH statement has been processed. Also, the following statement is generated:

FINISH: ON FINISH SYSTEM;

Then each entry in the precompiler's file table is analyzed. If the file was used inconsistently in the program, a diagnostic comment is generated and the next file analyzed. Otherwise, a Mainline File Control Block (MFCB) declaration is generated, using the file-name as the major structure name and as the initial value of the title. Any file attributes implied by the usage of the file are generated into the initial value of the filename.ACCESS and filename.FUNCTION fields. Statements are generated to "automatically" CLOSE the file, just the same as for a CLOSE statement.

After all files have been analyzed, the following statement is generated:

RETURN;

In all programs, a declaration of the entry points to the Data Base Executive (DBPAC) is generated.

In all cases, a summary diagnostic comment is generated giving the number of DB diagnostic error comments in the program. This completes the FINISH statement routine specification.

F. CODING SPECIFICATIONS

1. Source Language

The DB preprocessor function procedure is coded using the preprocessor PL/I statements permitted in preprocessor PL/I procedures.

Statements to be INCLUDEd or generated into the intermediate source program are coded using PL/I.

2. Suggestions and Techniques

The DB preprocessor function procedure is coded in

a modular manner so that the syntax analysis of the argument is separate from the generation of statements. This modularity will allow much of the DB coding to be usable for any other extensions to PL/I that may be designed, such as a Terminal Support PL/I language extension.

The coding of the specific statement routines are "table-driven" where possible to facilitate any future changes in the generated text for a particular statement.

TOPIC B.2 - DATA BASE EXECUTIVE EXECUTION PROCESSOR

A. MODULE NAME

Data Base Executive Execution Processor
Program-ID : NDBMPAC (and NDBREAC)
Module-ID - DBPAC
Procedure Entry Point (control section name): #FIELD
Other Entry Points - #XREF,DBPACFR,DBPACFP
DBPACPF,DBPACFV,DBPFLDT

(NOTE: This specification is for the module NDBMPAC which contains INPUT and UPDATE capabilities. Module NDBRPAC is a subset of this module with only the INPUT functions.)

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

DBPAC executes all data base input/output for mainline programs.

Mainline PL/I programs are written with DBPL/I statements for data base input/output. (See the DBPL/I User's Guide, Section 8, Topic B.2). These statements are processed during compilation and CALL statements are generated (according to the DBPL/I-DBPAC Interface Specification, Section 3, Topic B.2). The CALL statements pass control to the various entry points in DBPAC. The first parameter passed in a CALL to DBPAC is a Mainline File Control Block (see Section 3, Topic B.4). Other parameters are passed for particular purposes as needed.

DBPAC executes the request indicated by the operation code in the MFCB. For physical input/output operations it CALLs appropriate entries in the RDBTSSIO module. Whenever DBPAC detects either a logical error or a physical input/output error it posts an error code in the MFCB. (See DBPAC Error Codes, Section 3, Topic B.3).

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

The DBPAC module does not do any terminal input/output or print any reports.

2. Input Data Sets

In INPUT mode there are no output data sets.

In INPUT or UPDATE mode a data base is the input to DBPAC. Its descriptor data set is always read in as part of the OPEN processing and its anchor and/or associate and/or subfile and/or inverted index data sets are read depending on the operations requested by the mainline program. (See Dataplex Descriptor File, Section 3, Topic B.7).

In OUTPUT mode only the descriptor data set of the data base is read as input to DBPAC (during OPEN processing).

When a mainline program is accessing the descriptor data set of a data base, "descriptor descriptor" tables coded in DBPAC are used instead of an input descriptor data set.

3. Output Data Sets

In UPDATE or OUTPUT mode a data base is the output from DBPAC. Its descriptor data set is updated as part of OPEN and CLOSE processing (setting and reseting the MNTNABLE and MNTNING switches). Its anchor and/or associate and/or subfile and/or inverted index data sets are updated or output depending on the operations requested by the mainline program.

4. Reference Tables

   a. DBPL/I - DBPAC Interface (see Section 3, Topic B.2)

   b. DBPAC Error Codes (see Section 3, Topic B.3)

   c. Mainline File Control Block (see Section 3, Topic B.4)

   d. List Structure (see Section 3, Topic B.5)

   e. Dataplex Descriptor File (see Section 3, Topic B.7)

   f. Inverted Index Format (see Section 3, Topic D.5)

g. FLDTAB Table (see Section 3, Topic F.10)

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Receive Control

DBPAC receives control by being called from a "mainline" program. The entries at the beginning of the module are described here; entry DBPFLDT is described in paragraph "f" below. All entries receive a Mainline File Control Block (MFCB) as their first parameter. DBPAC treats the MFCB as a simple parameter; that is, DBPAC does not know that the MFCB is a CONTROLLED structure allocated by the mainline: DBPAC never ALLOCATEs or FREEs an MFCB.

For the #FIELD and #XREF function entries, an appropriate operation code is posted in the MFCB and the second parameter, which is a file name, is copied into the MFCB. This is necessary because the function references in the mainline have not been expanded by the DB preprocessor. (At the other entry points, which are for DBPL/I statements, the operation code and, if appropriate, the field name have been posted in the MFCB by statements preceding the CALL in the mainline. These assignment statements were generated by the DB preprocessor.)

The DBPACFR entry handles a user record in the form of a character string as its second parameter.

The DBPACFP and DBPACPF entries both handle a user list pointer as their second parameter. DBPACPF additionally accepts a user subscript as its third parameter. A switch indicating the absence or presence of a user subscript is set.

The DBPACFV entry handles a user field value in the form of a varying length character string as its second parameter. The DBPACFV entry is also used for all statement calls

that only pass an MFCB without a second parameter.

b. Common Code

Handling for PL/I errors that may occur in DBPAC is initialized so that they will cause a jump to paragraph "m" below before returning to the mainline.

If the MFCB is closed and a redundant CLOSE operation is attempted then control branches directly to the common return paragraph "m". If an OPEN operation or an operation that can imply opening (most record level operations) is encountered then control branches to the open routine - paragraph "d". If the operation can not imply opening then an error is raised: a specific error code is posted in the MFCB and control jumps to the common return - paragraph "m". This is an example of the general method DBPAC uses when it detects an error.

If the MFCB is open the operation code is checked for validity. Close and open (which is re-open in this case) operations branch to the close routine-paragraph "c". Record operations branch to paragraph "e". Get operations branch to paragraph "h". Put and Reput operations branch to paragraph "i". An invalid operation code raises an error and jumps to the common return - paragraph "m".

c. Close Routine

The close routine is used for the close operation and also for the open operation in re-open cases when by implication the dataplex must be closed first. For each data set in the dataplex the unlock subroutine is called and the ASMCLOS is called. (The Unlock subroutine is also used by the record level operations. It writes or rewrites or releases any new or modified or locked record by calling ASMPUT or ASMPUTK or ASMREL).

If a non-descriptor file was output or updated then the MNTNABLE or MNTNING switch in the anchor file descriptor record is updated by calling ASMDCB, ASMFNDS, ASMOPEN, ASMGETK, ASMPUTK and ASMCLOS on the descriptor data set.

For a simple close operation control branches to the common return. For an open operation control branches to the open routine - paragraph "d".

d. Open Routine

The open routine is used for the open operation and also when record level operations imply opening. The title value in the MFCB is dollar sign padded. For implied open operations, implied access and mode attributes are assigned. The master userid is obtained by calling ASMID. For a re-open operation on the same dataplex with the same security password, the following descriptor read-in and File Control Block (FCB) initialization steps are bypassed. For an open operation on a descriptor data set, a pointer to the hard-coded descriptor descriptor table in main storage is posted in the MFCB and the following descriptor read-in step is bypassed.

To read in the descriptor records, ASMDCB, ASMFNDS, and ASMOPFN are called. Then for each region (describing one date set) ASMGETK is called to read the file descriptor record and ASMGET is called repeatedly to read the field descriptor records. The file descriptor's DESCRCT field governs the size of the adjustable DESC table to be allocated and how many field descriptors to read. The applicable RSECTYCD record security mask is obtained from the file descriptor or a null mask is assumed. As the field descriptors are read in, they are checked for valid format. A descriptor for the RECLEN field is bypassed except on the first data set. When the descriptor for the key field is found, it is stored at the top of the DESC table, other descriptors are stored sequentially which is alphabetical by field name. For INPUT mode, field level SECURITY codes are checked: failure results in the field descriptor being bypassed and any dummy descriptor for the field having its field name nulled. Any superfield descriptors encountered are counted for checking later. For the first data set (the one specified by the TITLE clause) references to associate, subfile, and/or index files are tabulated to govern which other descriptor regions are to be read

in. After all the relevant descriptor regions have been read in, any superfield descriptors are reread (by calling ASMGETK) so that their component fields may be checked (if a component has failed security checking then the superfield also fails). Finally ASMCLOS is called to close the descriptor data set.

The main storage descriptors built during OPEN are telescoped; i.e. they contain no dummy descriptors. When associate and subfile descriptors are read, only the header and key field descriptors are posted to the DESC table for the data set. For each field descriptor, the corresponding dummy field descriptor already in the anchor DESC table is found and overlaid.

Next for each data set a File Control Block (FCB) is allocated and a skeleton Data Control Block is copied into it and ASMFNDS is called. For OUTPUT or UPDATE mode a null record is composed in the FCB by finding the primary field descriptors. After the FCBs are all initialized, then file and field subscripts (INVFLCUR,ASSOCCUR,SUBFLCUR, and RELFLDSS) are determined once for all to save the need for search loops elsewhere in DBPAC.

If a non-descriptor file is being opened for output or update then the MNTNABLE or MNTNING switch in the anchor file descriptor record is updated by calling ASMFNDS, ASMOPEN, ASMGETK,ASMPUTK and ASMCLOS. The first data set (the one specified by the TITLE clause) is always opened by ASMOPEN being called. If it is a descriptor data set, it is positioned to the specified region by calling ASMSTLK.For output or update, any subfile data sets in the dataplex are opened and the highest id-key in use is found in the descriptor header record for the subfile, DESC.HDR.ID_KEY_REG. If the operation was an explicit open then control branches to the common return. Otherwise it was an implicit open and control proceeds to the record routine.

e. Record Routine

The record-level routine is used for WRITE,

LOCATE, READ and UNLOCK operations. The WRITE operation is handled separately by calling ASMPUTK and branching to the common return.

If the statement has a SUBFILE or an INDEX clause then the field name is found in the DESC table to determine the data set for the operation. For output or update, the Unlock subroutine is called for the particular data set or for the anchor and associate data sets. (The Unlock subroutine is also used by the close routine. It writes or rewrites or releases any new or modified or locked record by calling ASMPUT or ASMPUTK or ASMREL.) For any UNLOCK operation control branches to the common return.

For LOCATE and READ operations, the element GET cursors are reset for the particular data set or for the anchor and associate data sets. The LOCATE SUBFILE operation is handled separately at this point: control branches into the Put routine to find the anchor or associate control field for the subfile. A subrecord id-key is determined from the highest id-key in the control file or, if it is null, from the highest id-key previously used in the subfile. This highest id-key is found in the header descriptor for that subfile, DESC.HDR.ID_KEY_REG. A current subrecord is built by copying the null record built by the open routine, posting the new id-key and posting the parent key field by copying the anchor key field. Control branches into the Put routine again to put the new element into the control field. To better ensure dataplex integrity, the anchor or associate record containing the control field is immediately written or rewritten and reread by calling ASMPUTK and ASMGETK. If the control field is on an associate and the anchor record was newly located then it is written and reread too. The LOCATE SUBFILE operation is thus complete and control transfers to the common return.

If a particular subfile or index data set has not been opened by this point (for a READ operation) it is opened by a call to ASMOPEN.

Since the highest id-key is maintained in the

subfile descriptor header record, whenever a LOCATE SUBFILE is performed the subfile header record is re-written at CLOSE time.

An anchor LOCATE operation is handled separately at this point: control branches to the Validate key routine (described with and also used by the READ KEY operation) and then an attempt is made to read the new key using ASMGETK. If the new key is found, the record is made current (just as if a READ KEY operation had been requested) and an error is raised. Normally, the new key will not be found and a current anchor record is built by copying the null record built by the Open routine (or, for a descriptor data set, a hard-coded null file or field descriptor record is copied) and the new key valve is posted in it. The LOCATE operation is thus complete and control transfers to the common return.

The remainder of the Record routine processes the various READ operations. Spanned index reads are handled separately at this point: their fundamental objective is to make the last record of a spanned region current. For read INDEX forwards DEPAC attempts to read with a suffix of 00 hex. If successful, forward sequential reading is done until the end of the region is passed. Then, a direct read by key is made using the last suffix encountered for the region. For read INDEX KEY the validate key routine (described later) is used. Then for all types of read INDEX, ASMGET is called to read the last record of the new region. The read spanned INDEX operation is thus complete and control transfers to the common return.

Normal (un-spanned) reads are processed as follows. For read forwards, it is unnecessary to do any file positioning. For read PER SUBFILE, the parent key value is taken from the current subrecord for use without validation. For read by KEY, the Validate key routine is used. The Validate key routine (also used for LOCATE KEYFORM) calls the generic conversion routine, if specified in the key field descriptor, and then calls the validation routine, if specified, using "CALL CALL" service for both purposes. For read LIST, the appropriate key

value is taken from the list (next forward, next backward, or by subscript) for use without validation. Then for all non-locking direct reads (PER SUBFILE, by KEY, or from LIST), ASMSTLK is called to position to the desired record. Now the file is positioned for all reads (except direct locking) and ASMGET is called to actually read the desired record. Then if the record is to be locked and for direct locking reads, ASMGETK is called to reread or read the record and lock it for exclusive use. Next, for INPUT modes, any record level security checking is done; if it fails and it was a sequential read (forwards or backwards), control loops back to do another sequential read until a record that passes security or end-of-file is encountered. If record security fails for a direct read, a key-not-found error is raised. For reading a descriptor data set only, the region is compared to determine if the read stayed within the region that was opened and the key is checked to determine if a file or a field descriptor was read so that the pointer to the appropriate hard-coded descriptor descriptor table can be posted in the MFCB to govern subsequent field level operations. If an anchor record was read, then all subfiles are checked: any having a current subrecord with a different parent key to the new anchor key are marked "not current". If a subfile record was read, then the anchor and all other subfiles are checked: any having a current (sub) record with a different (parent) key to the new subrecord's parent key are marked "not current". Normal (un-spanned) read operations are thus complete and control transfers to the common return.

f. DBPFLDT Entry

The DBPFLDT (Post FLDTAB) entry is provided to build a Field name Table by reference to DBPAC's main storage descriptor tables built by the Open routine. This entry is not supported by a DBPL/I statements a mainline program must:

i. execute a DBPL/I OPEN statement or a record level statement implying opening.

ii. "CALL DBPFLDT(mfcb); where mfcb is the file name of the dataplex that was opened.

iii. have a "% INCLUDE IISRMAC(FLDTAB);" statement to copy in the declaration for FLDTAB. Use of this entry is optional; DBPAC makes no use of FLDTAB.

g. FLDTAB Routine

The FLDTAB routine is entered only from the DBPFLDT entry described above. FLDTAB is allocated or freed and reallocated with its size adjusted to hold the number of field names in the dataplex. The data base name is posted without trailing dollar signs. The FLDTAB routine then sets up the main storage descriptor table linking a pointer chain, DESC_FLD.FCP, starting with the anchor key descriptor connecting all fields, except RECLEN, in the pre-defined format 4 order. SEQ_FORMAT.# in FLDTAB is posted with the field counts for formats 1 through 5. Retrieval field name functions are supported with the DBFIDU module. In addition, the DESC_FLD.GROUP indicator is posted as '1', '2', or '3' to show the field is available from the anchor alone, from one or more associate records, or from a subfile record, respectively. The FLDTAB routine is thus complete and control branches to the common return.

h. Get Routine

The Get routine is used for all GET operations and for the #FIELD and #XREF functions.

When a field name has been passed or posted in the MFCB, it is found in the DESC table to determine the data set for the GET; otherwise, the first data set (the one specified by the OPEN TITLE clause) is implied.

If that data set does not have a current record, then for the #XREF function a zero is returned. If it is the anchor data set and any subfile has a current record, its parent key will be used to read (using ASMGETK). The anchor record whose record security will

be checked: if it fails, a null value will be returned and control branches to the common return or, for #FIELD, a zero is returned.

The GET RECORD operations is handled by copying the record from the FCB to the user's string and branching to the common return.

For the GET LIST SET statement and GET INDEX LIST SET statement the cross-reference field descriptor is found and control branches down to the Get Field routine.

For the GET KEY SET, GET SUBFILE KEY SET and GET INDEX KEY statements the appropriate key descriptor is found and control branches down to the Get Field routine.

For the #XREF function the cross-reference field descriptor is found and control branches down to the Get Field routine.

For GET FIELD, #FIELD and GET SUBFILE LIST SET if the descriptor found previously was a dummy, then the corresponding real descriptor must be found in an associate descriptor table. For GET FIELD and #FIELD of a superfield, a loop is initialized to take each component field, starting with the first, find its read descriptor and record (using ASMOPEN and ASMGETK for an associate record if necessary) and perform the Get Field routine repeatedly until the superfield has been composed or its count determined.

The Get Field routine uses the current record of the current data set and a direct field descriptor and its file descriptor to extract a field value. It handles a bit field, a fixed length byte field, a simple variable field, a fixed length element of a multi-element field or a variable length element of a multi-element field.

GET KEY SET operations are handled separately after the fixed length key has been extracted. The key value is posted and control branches to the common return.

GET LIST SET operations are handled separately after the multiple fixed length element field has been found. For the SUBFILE option the subfile id-key field name

is found in the subfile descriptor table. For an index option, if the index is spanned and the last suffix is greater than zero, the first record in the region is read using ASMGETK and control branches back to the Get Field routine. A list is posted with the whole multi-element field value. For a spanned index, if the suffix is less than the last in the region, then the next index record is read using ASMGET and control branches back to the Get Field routine; this repeats until the whole region has been copied into list segments and the data set is positioned at the last record of the region again. GET LIST SET operations are thus complete and control branches to the common return.

The #FIELD function is handled for the null and real value cases of all five types of direct fields and for the case of an empty associate data set or an absent associate record. Superfields are handled by effectively evaluating #FIELD for each component to determine the net count. The #XREF function for a spanned index calculates the number of cross-references on records preceding the last in the region by assuming full maximum length records and adds the number of cross-references on the last record. The #FIELD and #XREF functions are thus complete and return their function value directly (without branching to the common return).

The GET INTO operations are handled for the null and real value cases of all five types of direct fields and for the case of an empty associate data set or an absent associate record. When the external form of the field is needed and a reformatting routine is specified in the descriptor, it is called using "CALL CALL" service. Superfields are handled by looping back to get each component field and concatenating them together; if the superfield descriptor specifies a reformatting routine, it is called using "CALL CALL" service. The GET INTO operations are thus complete and control branches to the common return.

i. Put Routine

The Put routine is used for PUT and REPUT operations. The field name passed in the MFCB is found in the DESC table to determine the data set implicated. If it is an associate data set, it is opened, if necessary, by calling ASOPEN and read, if necessary, by calling ASMGETK and if the record is absent a current associate record is built by copying the null record built by the open routine and the anchor key value is copied into it. If a generic conversion routine is specified in the field descriptor, it is called using "CALL CALI" service. If a validation routine is specified, it is called using "CALL CALL" service. If an anchor key or a subfile id-key is being REPUT to null, then control branches to the Delete routine described in paragraph "j" below.

Using the current record of the current data set and a primary direct field descriptor and its file descriptor, a bit field, a fixed length byte field, a simple variable field, a fixed length element of a multi-element field has a new value PUT or REPUT into it. For a fixed length field or element, the new value is justified right or left depending on the NUMALIGN switch in the field descriptor. For a variable length or multi-element field, the field length and record length (RECLEN field) are adjusted as necessary. If the field is indexed and had a non-null value, then the Delete XREF subroutine (described in paragraph "l" below) is called. If the new value is non-null and the field is indexed, then the XREF subroutine (described in paragraph "k" below) is called. The PUT and REPUT operations are thus complete and control branches to the common return.

j. Delete Routine

The Delete routine is used when an anchor key or a subfile id-key is being REPUT to null.

Nulling a subfile id-key indicates that a subrecord is to be deleted. The subfile control field descriptor is found and if it is on an associate, the associate data set is opened, if necessary, by calling ASOPEN and read, if necessary, by calling ASMGETK. The control field element is found and excised and the field length and RECLEN are

decremented. Then the subfile descriptors are searched: for all indexed fields, the Delete XREF subroutine (described in paragraph "l" below) is called for each element value. Finally the subrecord is deleted by calling ASMDELR and control branches to the common return.

Nulling an anchor key indicates that an anchor record and its associated and subordinate records are to be deleted. The anchor descriptors are searched for subfile control fields and for indexed anchor or associated fields. If a control or indexed field is found on an associate data set, it is opened, if necessary, by calling ASMOPEN and read, if necessary, by calling ASMGETK. For each control field, the subfile is opened, if necessary, by calling ASMOPEN and each element is used to read a subrecord using ASMGETK. The subfile descriptors are searched for every subrecord: for all indexed fields, the Delete XREF subroutine is called. Each subrecord is deleted by calling ASMDELR. During the anchor descriptor search, when an indexed anchor or associate field is found, the Delete XREF subroutine is called. Finally the associated records and the anchor record are deleted by calls to ASMDELR and control branches to the common return.

k. XREF Subroutine

The XREF subroutine is called from the Put routine when a non-null value is PUT or REPUT to an indexed field. If INDEXEXT and a reformatting routine are specified, it is called using "CALL CALL" service to transform the value to the form in which it is to be indexed. The inverted index data set is opened, if necessary, by calling ASMOPEN. Then an index read is attempted using ASMGETK (with a suffix of zero if it is spanned). If the record is not found, then the null record built by the Open routine is copied, the cross-reference and the indexed value are copied in, it is written by calling ASMPUTK and control returns to the calling program.

If an index record is found, then its highest (rightmost) cross-reference value is compared with the new cross-reference. If the new reference is lower, then the insertion point

is found by a binary search and the new reference inserted; otherwise the new reference is appended. If the index is not spanned or if the region only needs one record, the cross-reference field length and RECLEN are incremented, the index record is rewritten using ASMPUTK and control returns to the calling program.

In a spanned index region when the zero suffix record is full, if its last reference is less than or equal to the new reference then it is released by calling ASMREL; otherwise the insertion point is found by a binary search, the new reference is inserted, the last reference overflows to become the new reference to be propagated forward, and the record is rewritten using ASMPUTK. The suffix is incremented and control loops back to attempt a read of the next record of the region. This continues as long as full records are found. finally a short record is found to append to or a fresh record is created and the process is completed like a non-spanned case and control returns to the calling program.

1. Delete XREF Subroutine

The Delete XREF subroutine is called from the Put routine when an indexed field that had a non-null value is being REPUT. It is also called exhaustively by the Delete routine for indexed fields. If INDEXEXT and a reformatting routine are specified, it is called using "CALL CALL" service to transform the value to the form in which it was indexed. The inverted index data set is opened, if necessary, by calling ASMOPEN. If the index is spanned, the index region is read forward to find the last record in the region. Whether or not the index is spanned, ASMGETK is called to read the index record (with the highest suffix if it is spanned). If the index is not spanned or if the region only has one record, then the cross reference is found by a binary search and excised, the cross-reference field length and RECLEN are decremented, the index record is rewritten using ASMPUTK and control returns to the calling program. In the exceptional case of the index record only having the one cross-reference, it is deleted using ASMDELR

and control returns to the calling program.

In a spanned index region having more than one record, the lowest (leftmost) cross-reference value is examined before the binary search. If it is greater than the cross-reference to be deleted, then the whole cross-reference falls off to be rolled backward in the region. The record is then rewritten (with the field length and RECLEN decremented if necessary) using ASMPUTK or deleted using ASMDELR. Then the previous record is read using ASMGETK with the next lower suffix and the lowest cross-reference examined. This process repeats rolling one cross-reference backward in the region until the record is found with a lowest cross-reference less than or equal to the one to be deleted. The cross-reference is found by a binary search and excised, the rolled cross-reference from the record just processed is posted at the right end, the record is rewritten using ASMPUTK and control returns to the calling program. If the cross reference is not found on the record, it belongs on then the record is released using ASMREL and in the simple case control returns to the calling program; the intent has been accomplished. However, if rolling back had been started in a spanned region, one cross-reference is still in limbo, so control branches into the XREF subroutine which will roll one cross-reference forward from that point to reconstruct the region before returning to the calling program; this should be an extremely infrequent occurrence.

m. Return

The common Return is used by all routines. The only exception is that when the #FIELD or #XREF functions complete successfully they return directly.

When an error has been detected, an error code is posted in the MFCB. The address of the MFCB is posted in DBEFCBP to assist any mainline having multiple MFCBs. If the mainline has a current DBPL/I ON ERRORFILE GO TO ... action, then RETRNPT is called to post MFCB.ONRETURN and DBPAC is left by branching to the mainline label in MFCB.ERROR.ROUTINE. Otherwise DBPAC is left by signalling the

PL/I ERROR condition which, unless the mainline catches it, will terminate the mainline program.

Normally, DBPAC is left by a simple RETURN statement and control returns to the mainline that called.

F. CODING SPECIFICATIONS

1. Source Language

DBPAC is written in PL/I. The DB preprocessor and DBPL/I are not used in DBPAC. Various Assembler language subroutines are used as mentioned in the Processing Requirements Narrative.

2. Suggestions and Techniques

When a desired field descriptor has been found by subscript in the tables, its address is held in a pointer variable and based structure references are used to avoid frequent re-evaluation of the subscript. Similar techniques are used whenever possible.

Binary search techniques are used to maintain the cross-reference lists in inverted index records in ascending sequence.

The facilities available in the DBDBIO module are used to the best possible advantage with the OS ISAM access method.

The DBPAC module is designed and implemented to be reentrant under multi-programming; automatic, controlled and based storage are used appropriately. One known exception is that the main storage descriptor descriptor tables are static for efficiency; if two or more users attempt to access the same descriptor data set region concurrently they may encounter interference on the multi-element field cursors (only RSECTYCD, NAMEFLD and SECURITY fields are affected).

82-83

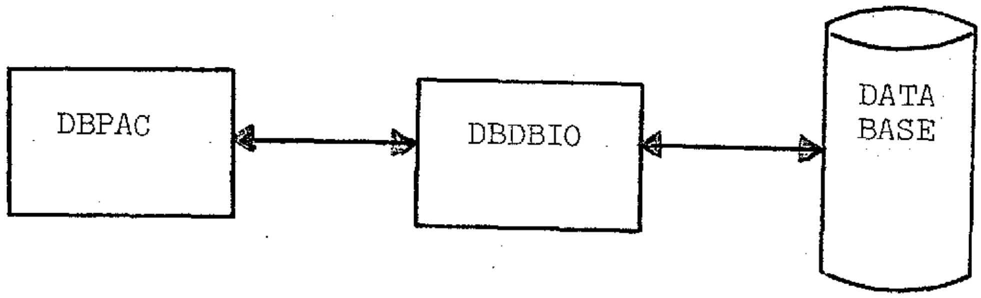


Figure 1. I/O Block diagram

TOPIC B.3 - EXECUTIVE ASSEMBLER PROGRAMS

A. MODULE NAME

Executive Assembler Program
Program-ID - NDBDBIO
Module-ID - DBDBIO

B. ANALYST

Connie D. Becker
Edward J. Scheboth, Jr.
Neoterics, Inc.

C. MODULE FUNCTION

This program works in conjunction with the Data Base Executive Program (NDBPAC) and provides the assembler language macros required to handle the input, output and updating of ISAM files, as well as the handling of error conditions.

These ISAM files are the files of a dataplex and the Data Base Executive will call the Executive Assembler Program when it needs an I/O operation performed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      All of the files which make up a dataplex could conceivably be input, including descriptor files. The only real restriction is that the files be ISAM.

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      Same as input files.

   b. On-line Terminal Displays

      Not Applicable

   c. Formatted Print-outs

      Not Applicable

   d. Punched Card Output Files

      Not Applicable

4. Reference Tables

   Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   This program is designed to handle the input and output functions for the Data Base Executive (NDBPAC). It deals strictly with ISAM files.

   The program is divided into many routines, and each of these routines has a unique function. The Data Base Executive (NDBPAC) calls these routines individually to perform the various functions which are required. Associated with each of these calls is the passing of the required parameters.

   The Data Base Executive is required to perform the various calls to the Executive Assembler Routines in a logical order.

   The abilities of these assembler routines are comprehensive enough to handle any situation which might arise in the Data Base Executive. This includes the abilities to: open files for input, output, or update; read the file sequentially, read the file by key, position the file to the beginning, or the next record; and close the file. For example, if the Data Base Executive were

required to open a dataplex in the update mode and process records, the sequence of calls would be as follows:

CALL ASMDCB (parameters)
establish the files DCB (data control block).

CALL ASMFNDS (parameters)
link the DCB with the JFCB (job file control block).

CALL ASMOPEN (parameters)
open the file.

CALL ASMGETK (parameters)
read a record by key.

CALL ASMPUTK (parameters)
rewrite the record.

CALL ASMCLOS (parameters)
close the file.

DBDBIO is called from the Data Base Executive (NDBPAC). If no errors are detected by the assembler routines, the error switch (one of the parameters) is set equal to zero upon return to NDBPAC and the return is to the specified 'Good' return address (one of the parameters). If an error is detected by the assembler routines, the error switch is set with the proper error code and the return is to the next sequential instruction in NDBPAC. The error codes will have the following values when an error occurs in ASMOPEN, ASMPUTK, ASMGETK, ASMGET, ASMPUT, ASMSETL, ASMESTL, ASMREL, ASMCLOS, ASMDELR or ASMSTLK:

a. 08 - key not found
b. 12 - sequence error
c. 28 - LRECL greater than MAX
d. 31 - position past end of D.S.
e. 44 - no space for record
f. 48 - invalid I/O area
g. 52 - invalid I/O request
h. 56 - duplicate record
i. 60 - DCB was closed
j. 64 - overflow record
k. 68 - uncorrectable I/O error
l. 72 - no key length specified

The assembler routines will add 100 to all of the above error codes prior to returning to the Data

Base Executive (NDEPAC). The end of data exit sets the error switch to 99. The error switch is a fixed binary half-word.

The PL/I calls for ASMOPEN, ASMPUTK, ASMGETK, ASMGET, ASMPUT, ASMSETL, ASMESTL, ASMREL, ASMCLOS, ASMDELR, and ASMSTLK are illustrated in Table 1.

The first parameter is always the DCB address (DCB means Data Control Block). The second parameter is the record area, except for:

a. The open (ASMOPEN) - in this instance, it is a one byte function code -
   I = input
   O = output
   U = update

b. The close (ASMCLOS)
   ESETL (ASMESTL)
   STLK (ASMSTLK)
   REL (ASMREL) - in these instances, it is a one byte dummy character (no meaning.)

c. The DELREC (ASMDELR) - in this instance, it is the key.

d. The SETL (ASMSETL) - in this instance, it is a one byte function code -
   B = beginning
   N = next

The third parameter indicates the routine to which return is made if there are no errors.

NOTE: The error switch parameter for the following routines must be preset.

a. ASMGETK - 01 (Read by key)
   00 (Read by key exclusive)

b. ASMPUTK - 01 if KT(Write)
   00 if KS(Rewrite)

The routines and their functions are as follows:

a. ASMFNDS: This routine obtains the DDNAME for an associated DSNAME and posts it in the DCB. The DSNAME must appear in the JCL for the job step currently being executed. "ASMFNDS" has an associated entry point

"INTFNDS" which does initialization for "ASMFNDS" for a particular jobstep. it should be called during initialization for the application. It is a simple calll of form:

CALL INTFNDS OR L R15,=V(INTFNDS)BALR R 14,R15

The parameters required for successful execution of the FINDDS are as follows:

1. The DS name (35 characters)
2. The DCB address
3. The owner's ID
4. The error switch (key length)

b. ASMERSE: This routine erases the direct-access storage for a data set. In addition, it will remove the entry for a catalogued data set from the catalog. The DSNAME passed is padded with blanks to 35 characters. If a stored data set is opened by many users concurrently, a particular user cannot erase that data set until every other sharer actively using that data set issues a close.

Once a user is the only currently active task using the data set, he may erase it regardless of whether he has closed it or not.

The parameters required are the DSNAME and the error switch.

NOTE: For both ASMFNDS and ASMERSE, the error switch upon return to the Data Base Executive is equal to zero only if no error occurred.

c. ASMOPEN: This routine connects the data set to the system by completing the DCB (containing the attributes), indicates the manner in which the data set is to be processed and positions the data set for processing. The address of the SYNAD routine (SYNADRTN) and the address of the EODAD routine (EODADRTN) are posted to the DCB. The address of the save area is also posted to the DCB.

The parameters are as follows:

1. The DCB address
2. The function code
3. The 'Good' return address
4. The error switch

d. ASMPUTK: This routine moves a selected record from a user specified area to an output buffer. The system then includes the record in the output data set by key. This operates in one of two modes: Rewrite (KS) or Write (KT). Write releases any page level interlocks set for the data set. The parameters are as follows:

   1. The DCB address
   2. The record area (address)
   3. The 'Good' return address
   4. The error switch (preset:
      0 means Rewrite (KS),
      1 means Write (KT)).
   5. The key (address)

e. ASMGETK: This routine obtains a selected logical records from an input data set and moves it to a user specified area.
   The parameters are as follows:

   1. The DCB address
   2. The record area (address)
   3. The 'Good' return address
   4. The error switch
   5. The key (address)

f. ASMGET: This routine obtains the next sequential record and moves it from an input buffer to a user specified area. The parameters are as follows:

   1. The DCB address
   2. The record area
   3. The 'Good' return address
   4. The error switch

g. ASMPUT: This routine has the same parameters as the ASMGET routine. However, instead of reading a record, it writes a record.

h. ASMSTLK, ASMSETL: These routines position a data set. The parameters for both routines are as follows:

   1. The DCB address
   2. The code : K (by key)

B (beginning)
N (next)

3. The 'Good' return address
4. The error switch
5. The key (address, for ASMSTLK only)

i. ASMESTL: This routine repositions the pointer to the beginning of the file. This is automatically done in ASMGETK, ASMSETL, ASMSTLK.

The parameters are as follows:

1. The DCB address

j. ASMDELR: This routine flags a record for deletion from an ISAM file. The parameters are as follows:

1. The DCB address
2. The key
3. The 'Good' return address
4. The error switch

k. ASMCLOS: This routine closes the file (ISAM).

l. SYNADRTN, EODADRTN: When an end of file or some error is detected during any of the routines in this program, these routines set the proper error code in the error switch and return control to the Data Base Executive for appropriate action.

m. ASMDCB: This routine takes the DCB created in this program and moves it to the user's specified area. The only parameter is the user specified area (address).

n. ASMXTR, ASMPASS, ASMMUST: These entry points simply transfer control to the MTT monitor to maintain linkage conventions.

F. CODING SPECIFICATIONS

1. Source Language

Unlike most other modules for the NASIS system, the Executive assembler program (DBDBIO) is written entirely in Assembly language.

2. Suggestions and Techniques

a. Special attention is paid to the linkage conventions of the current PL/I compiler.

b. The Data Base Executive, by design, is the primary user of this program. However, the program is written so that programs other than the Data Base Executive can use it.

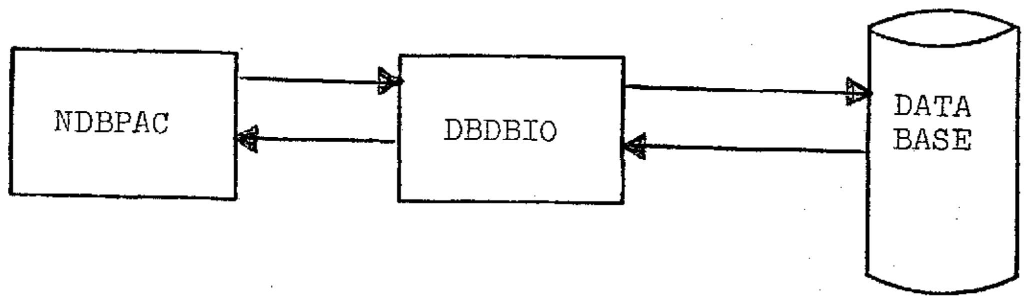


Figure 1. I/O Block Diagram

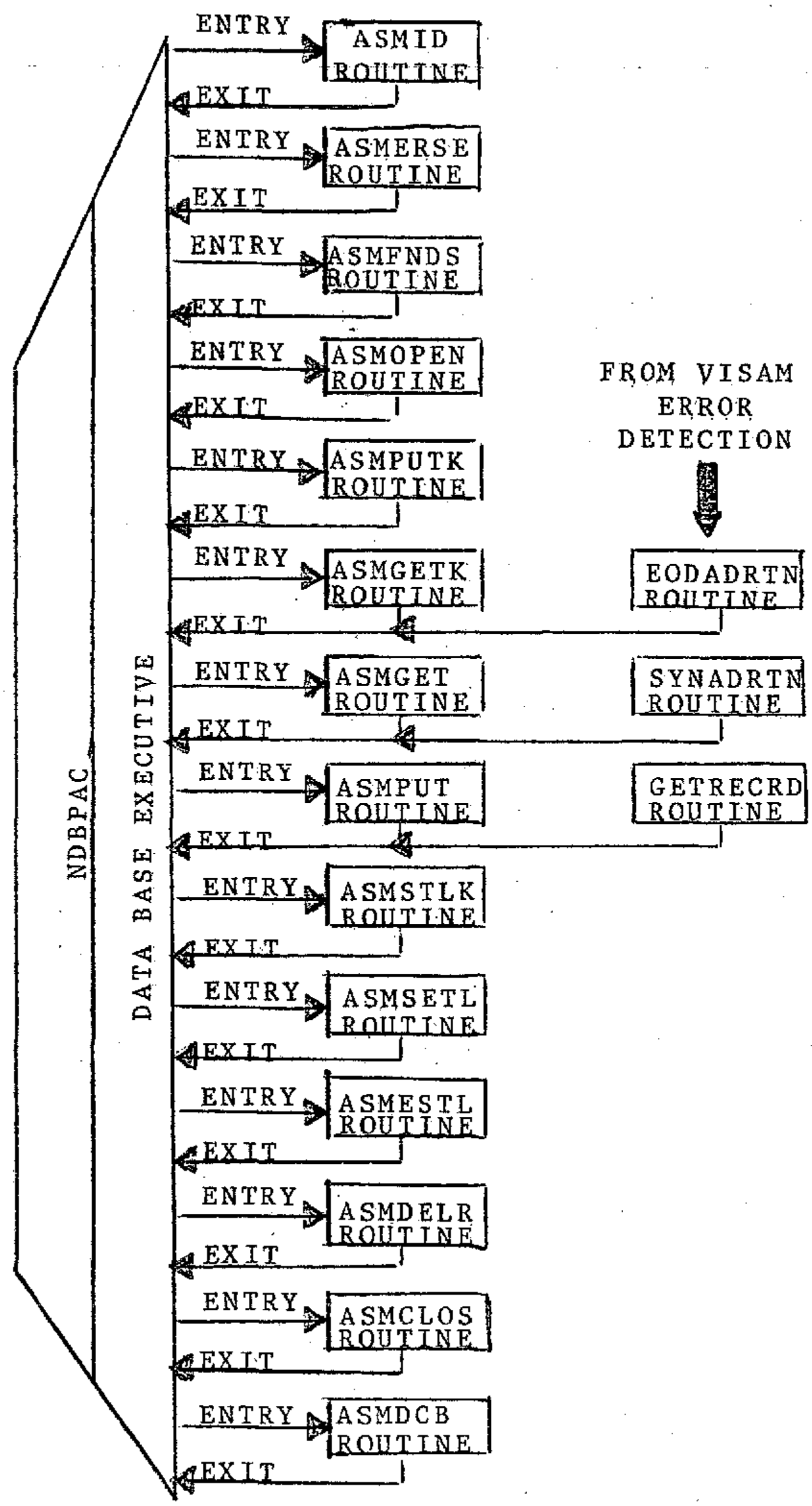


Figure 2 - Top level flow chart.

TOPIC B.4 - DATA BASE EXECUTIVE CONVERSION AND REFORMATTING ROUTINES

A. MODULE NAME

Standard Conversion and Reformatting routines for the Descriptor Editor and the Data Base Executive.

Program-ID - NDBEXITS
Module-ID - DBEXITS
Entry Points - See Table 1.

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

This module provides 31 standard general field conversion and reformatting routines. They are called by the Data Base Executive field processing routines (PUT, GET, and REPUT) if they are specified in the field descriptor record. The routines are written according to the DBPAC Exit Routines User's Guide (Section 8, Topic B.1) and may be used for user's database fields, if desired.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

The conversion routines (DBCVT__) are for use during PUT or REPUT field processing. They all accept a varying length character string argument and all allow the value to have leading and trailing blanks. They check the argument value according to the Notes in Table 1. If the argument value is invalid, they return with the BAD parameter left set. Otherwise they copy the value or convert it to the internal form and length shown in Table 1, reset the BAD parameter switch and return.

The reformatting routines (DBFMT__) are for use during GET field processing. They all accept a varying length character string argument (from the dataplex). If the argument length is not as shown under "Internal bytes" in Table 1, then the routine is being misused and the value "BAD. HEX=" is generated followed by the hexadecimal expansion of up to eight bytes of the argument. Normally the internal form of the value is reformatted to the external form and control is returned. These routines all produce exact length output (i.e. without leading or trailing blanks).

F. CODING SPECIFICATIONS

1. Source Language

PL/I with no DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

TOHC B.5 - LIST MANAGER

A. MODULE NAME

Program-ID: NDBLIST
Module-ID: DBLIST

B. ANALYST

George Oswald,
Neoterics, Inc.

C. MODULE FUNCTION

NDBLIST will be responsible fcr all requests for keys from an existing SET. It will contain the following functions in order to provide this facility. The DBPL/I statements handled by this module are as follows:

1. FREE LIST
   (a) General
   (b) Specific
2. GET LIST INTERNAL KEY INTO
3. GET LIST KEY INTO
4. GET LIST KEY(0)
5. GET LIST KEY SET
6. COPY LIST
7. LIST
8. #LIST
9. PUT LIST INTERNAL KEY FROM
10. SET LIST LIKE LIST

The Free temporary list and Unique list functions are called directly. The module will be called via PL/I conventions (DB Preprocessor Statements) and will use the facilities of SET Manager via PL/I calling conventions.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

   Not Applicable

2. Input Data Sets

   Not Applicable

3. Output Data Sets

   Not Applicable

4. Reference Tables

   a. SET CONTROL BLOCK (SCB)

   b. LISTERR

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

   1a. FREE LIST (General)

   ENTRY: DBPAC, see Figure 1a.

   1b. FREE LIST (Specific)

   ENTRY: DBPACT, see Figure 1b.

2. GET LIST INTERNAL KEY INTO

   ENTRY: DBGLIK, and

3. GET LIST KEY INTO

   ENTRY: DBGLKI, see Figure 1c.

4. GET LIST KEY(0)

   ENTRY: DBGLKO, see Figure 1d.

5. GET LIST KEY SET

   ENTRY: DBGLKS, see Figure 1e.

6. COPY LIST

   ENTRY: DUPLIST, see Figure 1f.

7. LIST

   ENTRY: LIST, see Figure 1g.

8. #LIST

   ENTRY: #LIST, see Figure 1h.

9. PUT LIST INTERNAL KEY FROM

   ENTRY: DBPLIK, see Figure 1i.

10. SET LIST LIKE LIST

   ENTRY: DESLLL, see Figure 1j.

11. FREE TEMPORARY LIST

ENTRY: DEFREET, see Figure 1k.

12. UNIQUE LIST

ENTRY: ULIST, see Figure 11.

13. Internal Routines

a. GET, see Figure 1m.

b. ERROR CONTROL, see Figure 1n.

c. FINISH LIST, see Figure 1o.

14. Narrative

NDBLIST provides an interface between a user's request for manipulation of a list and the SET Manager. The intenticn being:

1. Maintain the DBPL/I LIST statements as they are now defined by the NASIS system.

2. Isolate the manipulaticn of SET's to a single module (SET Manager).

3. Provide minimal amount of coding to support items 1 and 2.

In general, the NDBLIST functions will perform a minimal amount of data manipulation. This means that the requirements of each function are basically three:

1. Validation at Time of Entry

2. Call to SET Manager

3. Return Variables to Caller

The main exception is the LIST function which logically combines twc existing sets into a third set and returns an address of the new set.

The entry points to the NDBLIST will be as defined by DBPL/I. This retains the integrity of a user request (PL/I coded) for the NDBLISTM functions.

a. FREE LIST

A request to free a General or Specific list is an ERASE CALL to the SET Manager.

b. GET LIST INTERNAL KEY INTO

This entry requests a key via SET Manager from a current set (by set pointer) and returns to the user a variable string containing the key.

c. GET LIST KEY(0)

This will request via SET Manager that a current set's get cursor be reset to point to the first key within the set. This will be through a CLOSE CALL.

d. GET LIST KEY INTO

The entry will call SET Manager requesting a key from a current set (by set pointer). If the requested set indicates a conversion is necessary, the appropriate conversion routine will be called. Returned to the user will be a variable string containing the (converted) key.

e. GET LIST KEY SET

The entry will call SET Manager requesting a key from a current set (by set pointer). If the requested set indicates a conversion is necessary, the appropriate conversion routine will be called. Returned to the user will be a variable string containing the (converted) key.

e. GET LIST KEY SET

This entry will request a key from SET Manager and then write a key via SET Manager to a new or existing set. Returned to the user will be a pointer to the new list.

f. COPY LIST

Duplicates an existing list.

g. LIST

This entry allows a user to combine two existing sets by a logical operation:

()1 Logical OR, '|' (vertical). If the key is present in either set, the key is placed in the new set.

(2) Logical AND, '&' (ampersand). If the key is present in both sets, the key is placed in the new set.

(3) Logical NOT, '-' (minus sign). If the key is present in set 1 but not in set 2, then the key is placed in the new set.

It is possible for either the argument set (1) or the function set (2) to be NULL. The logical operation will return a pointer to either set or a NULL set.

Returned to the user is a set pointer to the new list.

h. #LIST

This entry requests the number of keys in a set and returns a variable containing the value. (Does not call SET Manager).

i. PUT LIST INTERNAL KEY FROM

This entry will obtain (a) key(s) from a user's variable and write it via SET Manager to an existing set.

j. SET LIST LIKE LIST

This entry will perform an OPEN CALL of a new set and return a set pointer to the user. The new set will have the same characteristics as the existing set.

k. FREE TEMPORARY LIST

This entry loops through the list chain and frees (calls SETERAS) for every list without its SCB.PERMEN bit on.

l. UNIQUE LIST

This entry creates a new list of non-duplicate keys from an input list. If no duplicates are found, the original list is returned as the new list.

F. CODING SPECIFICATION

1. Source Language

   This module will be coded in PL/I.

2. Suggestions and Techniques

   The LIST Manager will call the SET Manager via PL/I calling conventions. The structure LISTEBB will be used as the communicator between the SET Manager and the LIST Manager.

   An attempt will be made to maintain all structures and processing techniques so as to negate any changes to the modules requesting the use of the LIST functions.

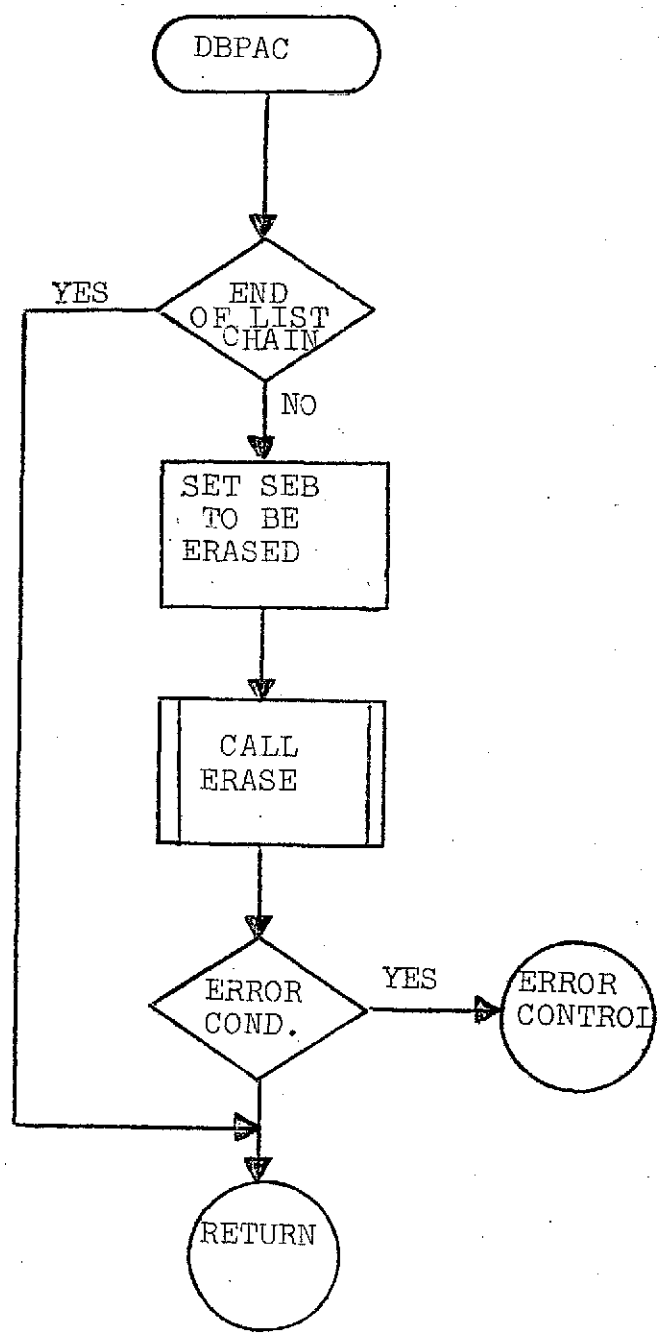


Figure 1a. DBPAC Entry

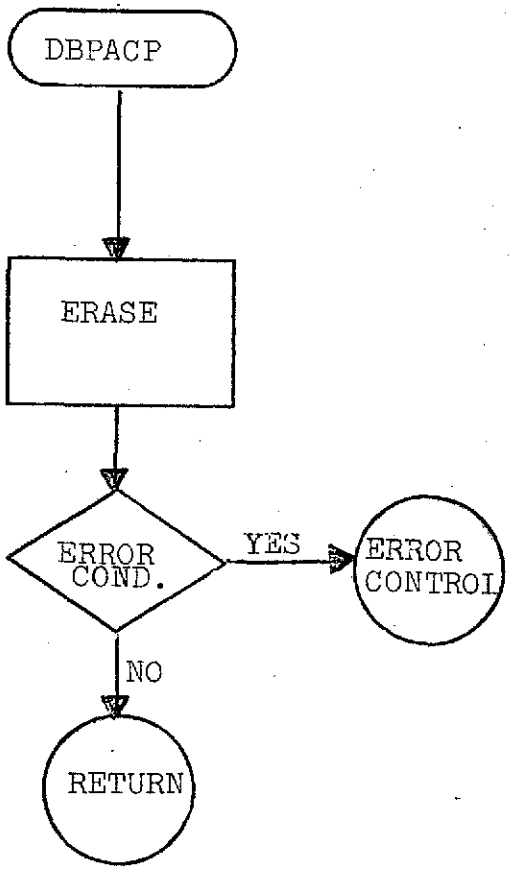


Figure 1b. DBPACP Entry

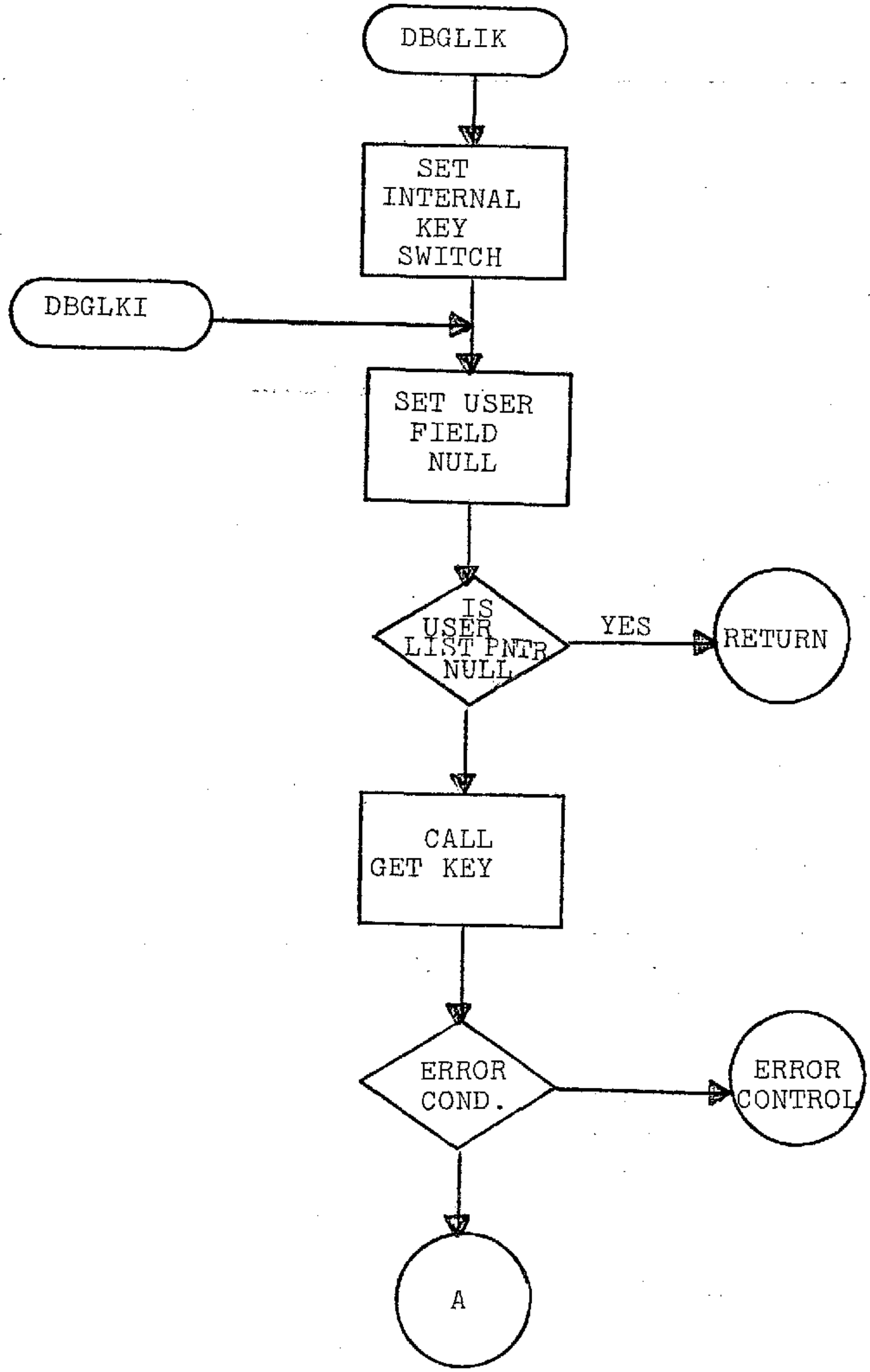


Figure 1c. DBGLIK and DBGLKI Entries (Page 1 of 2)

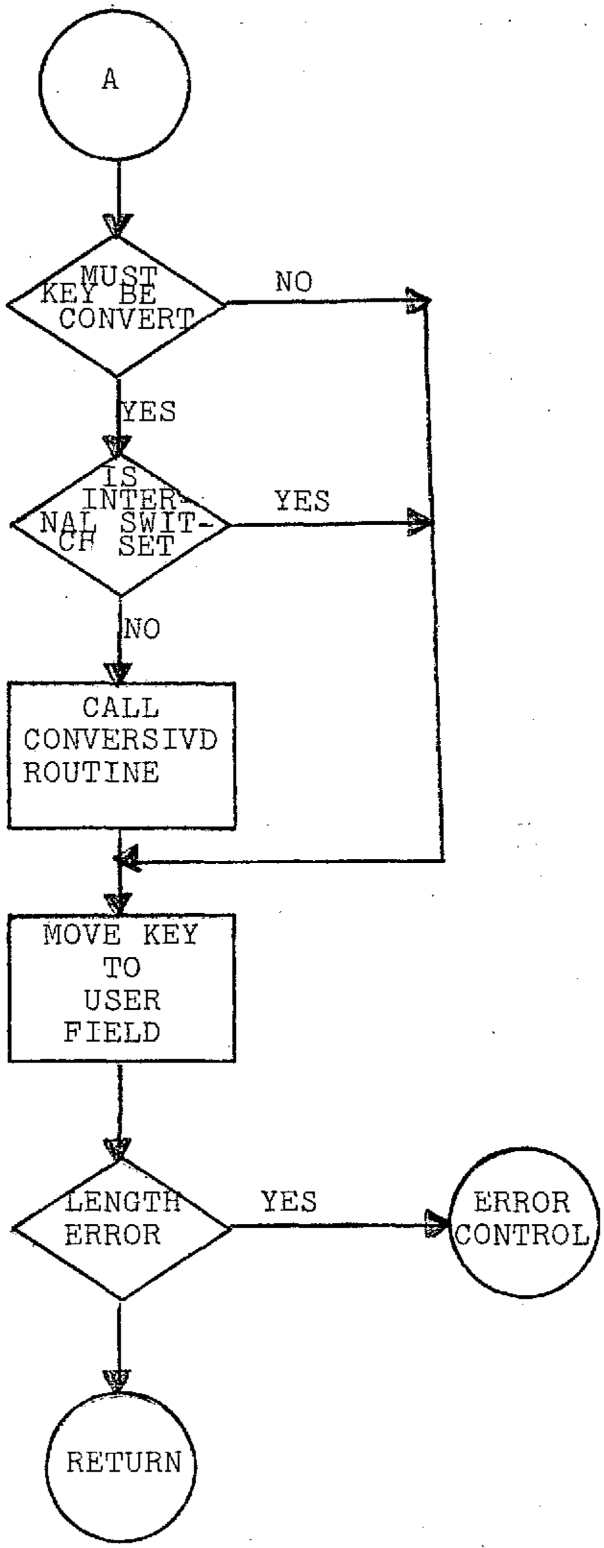


Figure 1c. DBGLIK and DBGLKI Entries (Page 2 of 2)

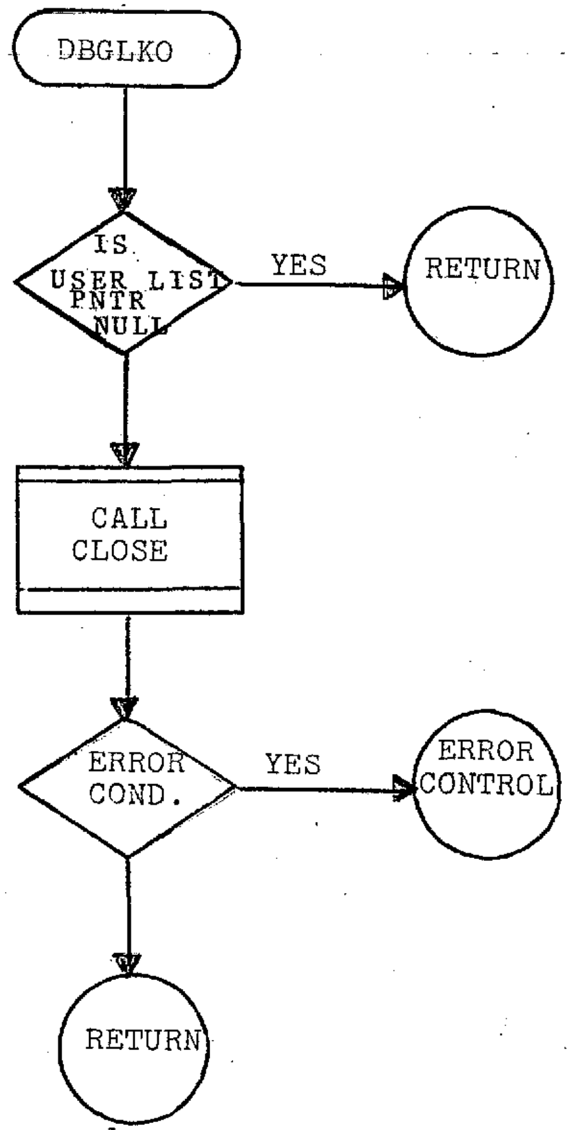


Figure 1d. DBGLKO Entry

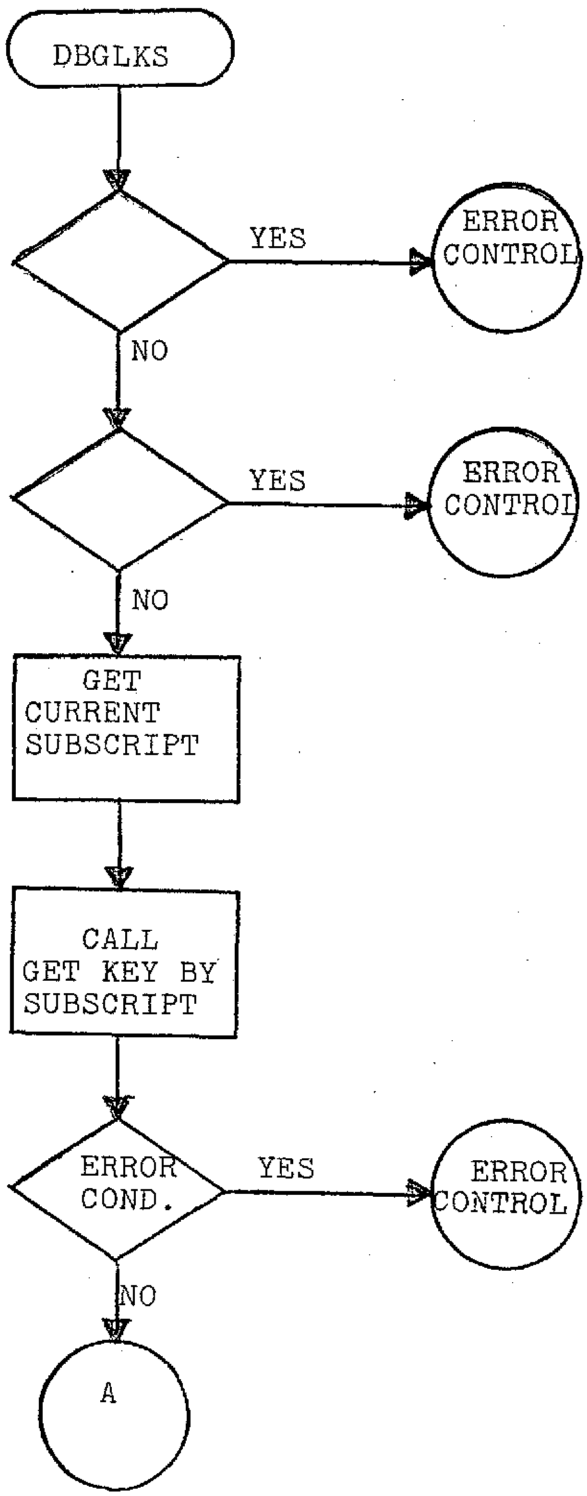


Figure 1e. DBGLKS Entry (Page 1 of 2)

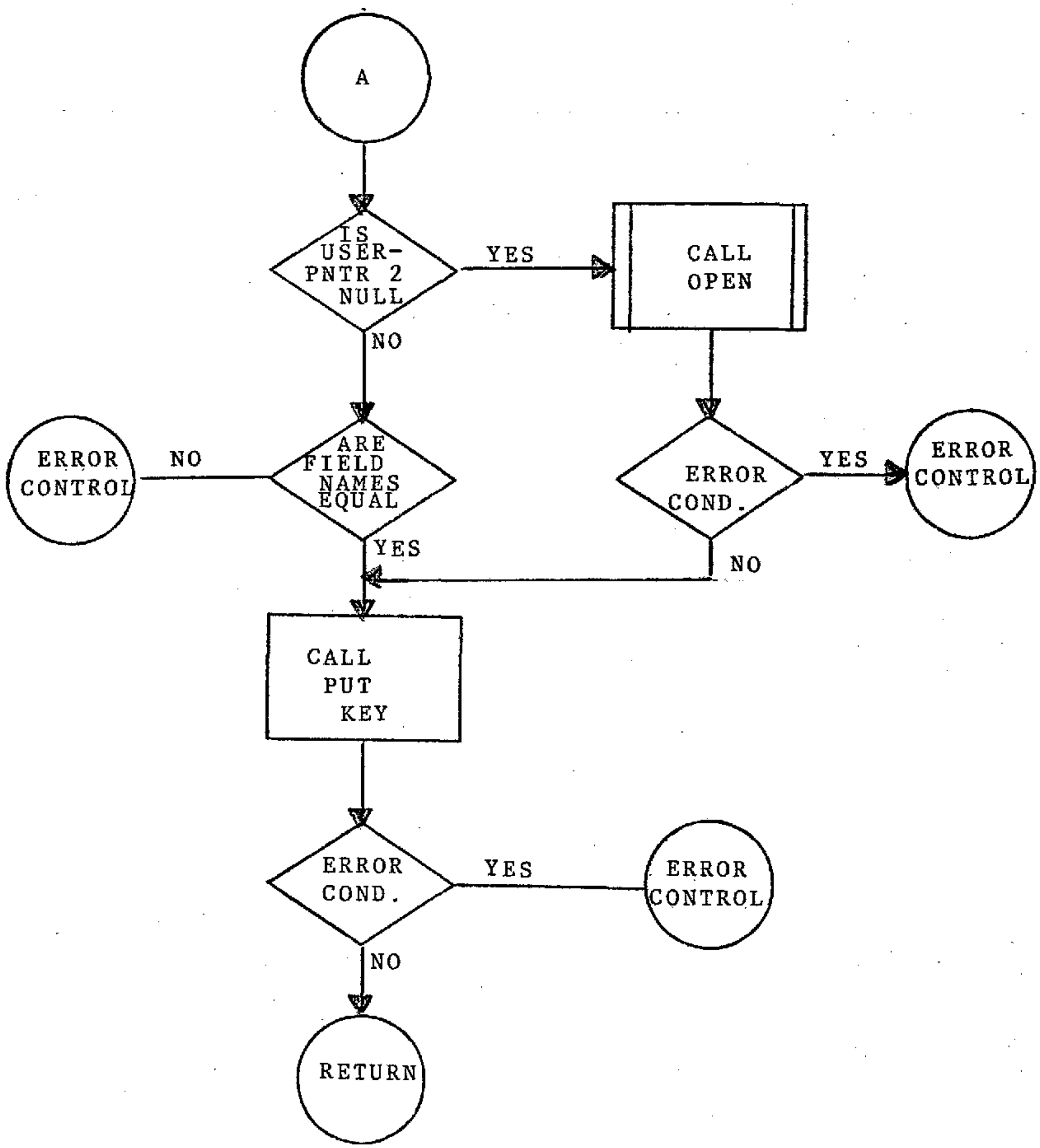


Figure 1e. DBGLKS Entry (Page 2 of 2)

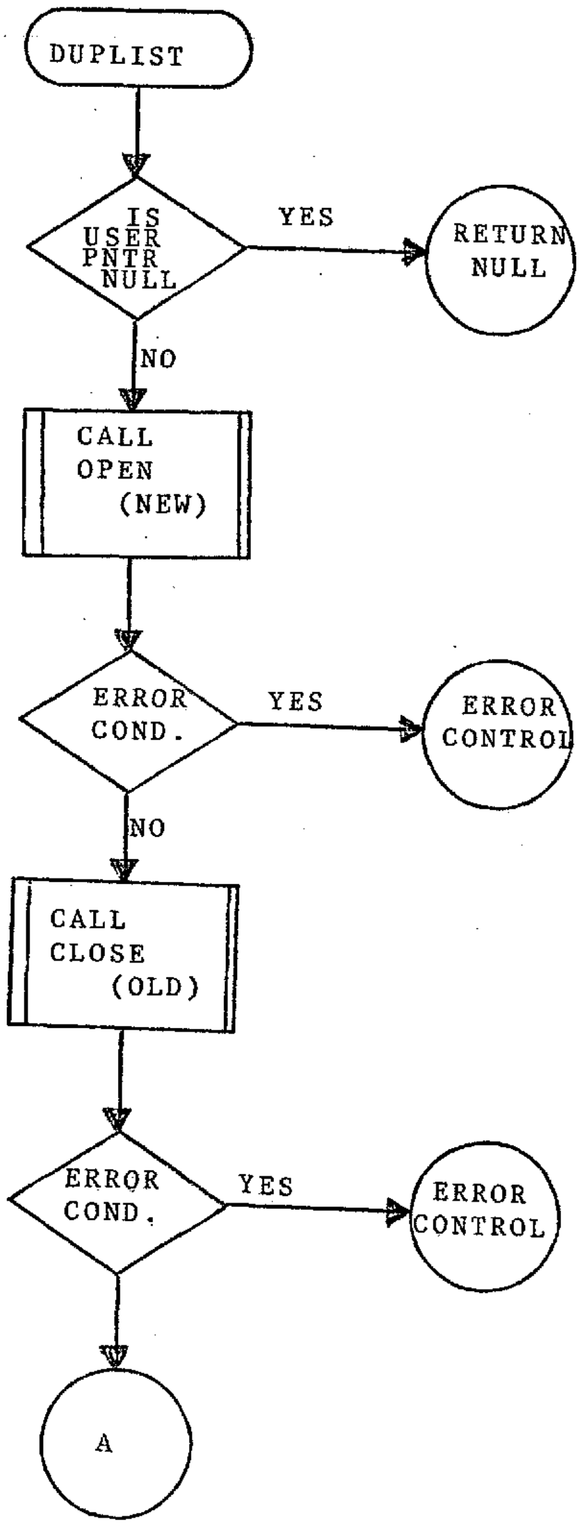


Figure 1f. DUPLIST Entry (Page 1 of 2)

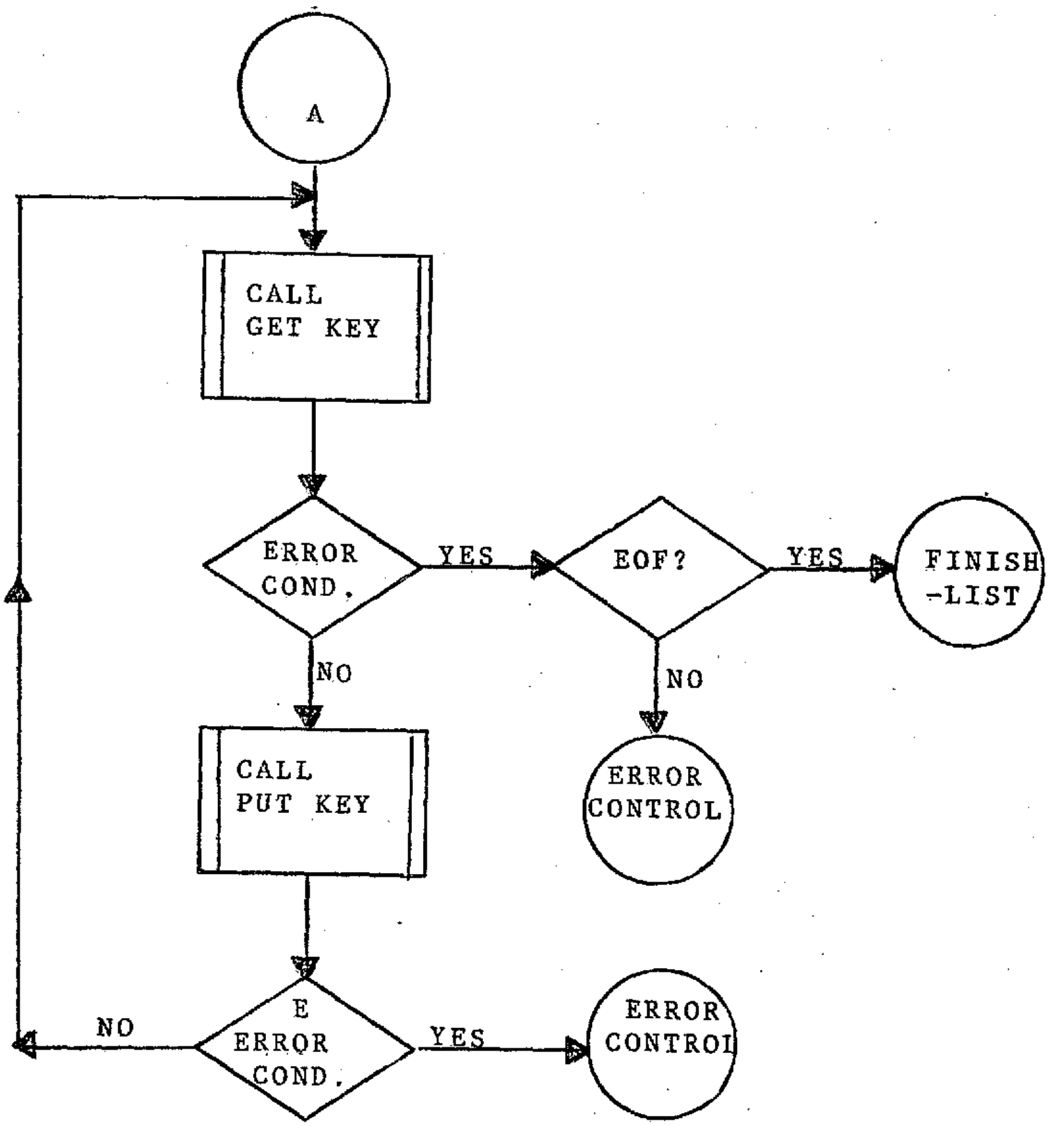


Figure 1f. DUPLIST Entry (Page 2 of 2)

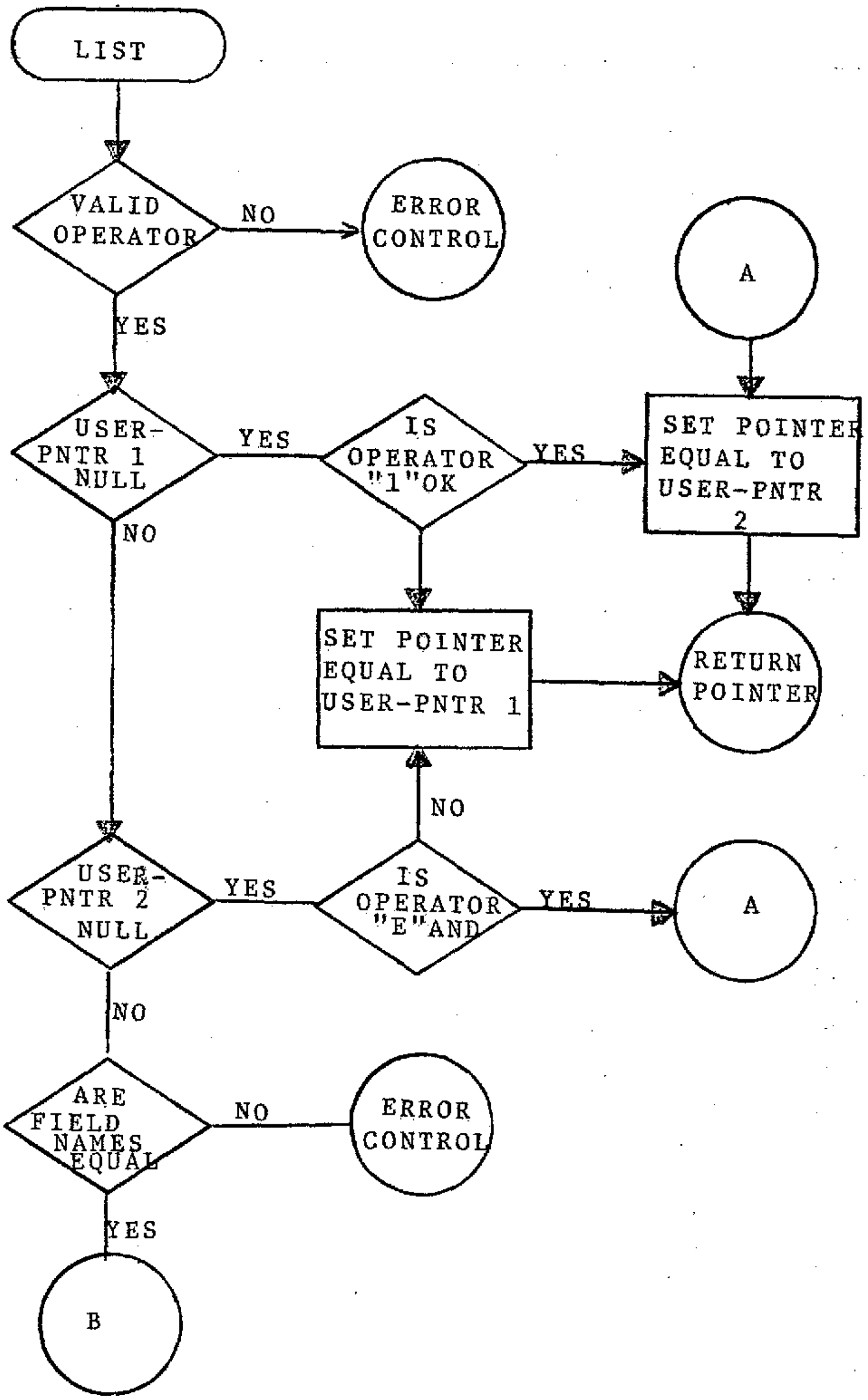


Figure 1g. LIST Entry (Page 1 of 4)

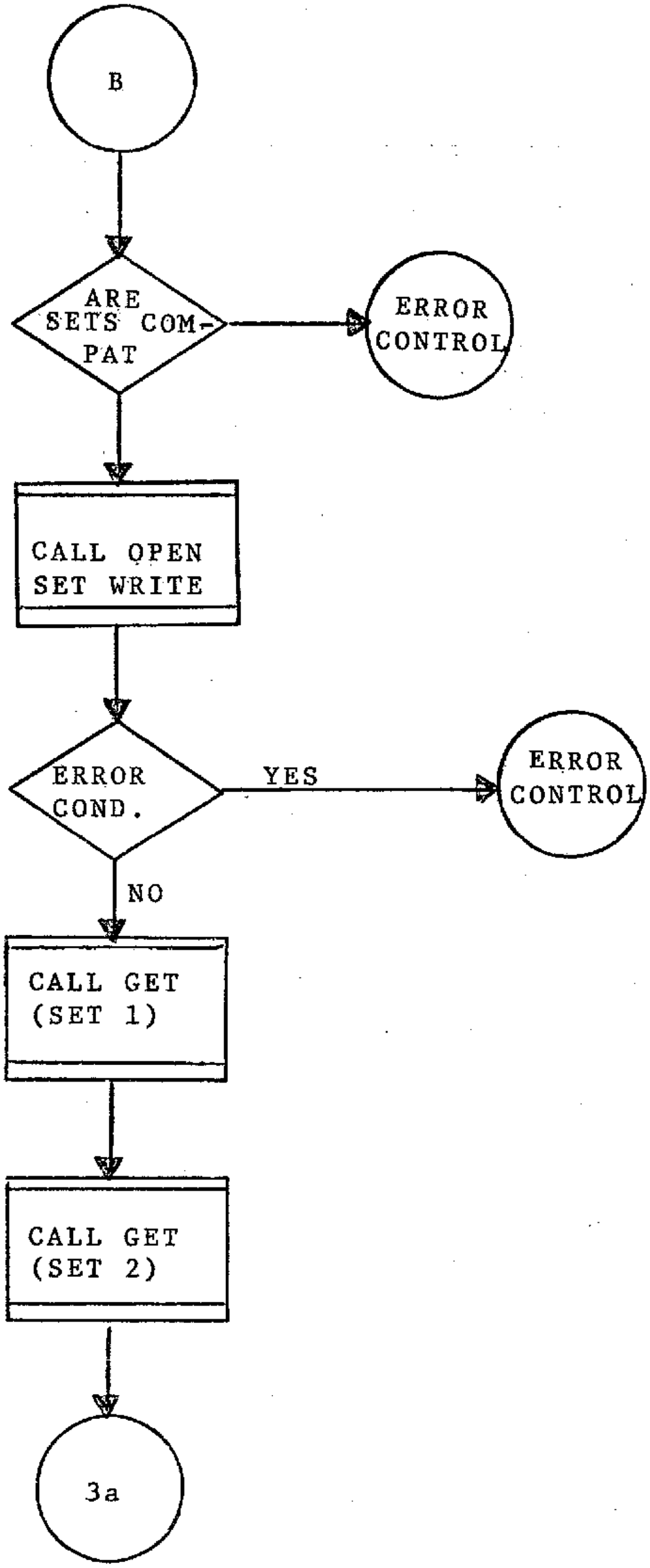


Figure 1g. LIST Entry (Page 2 of 4)

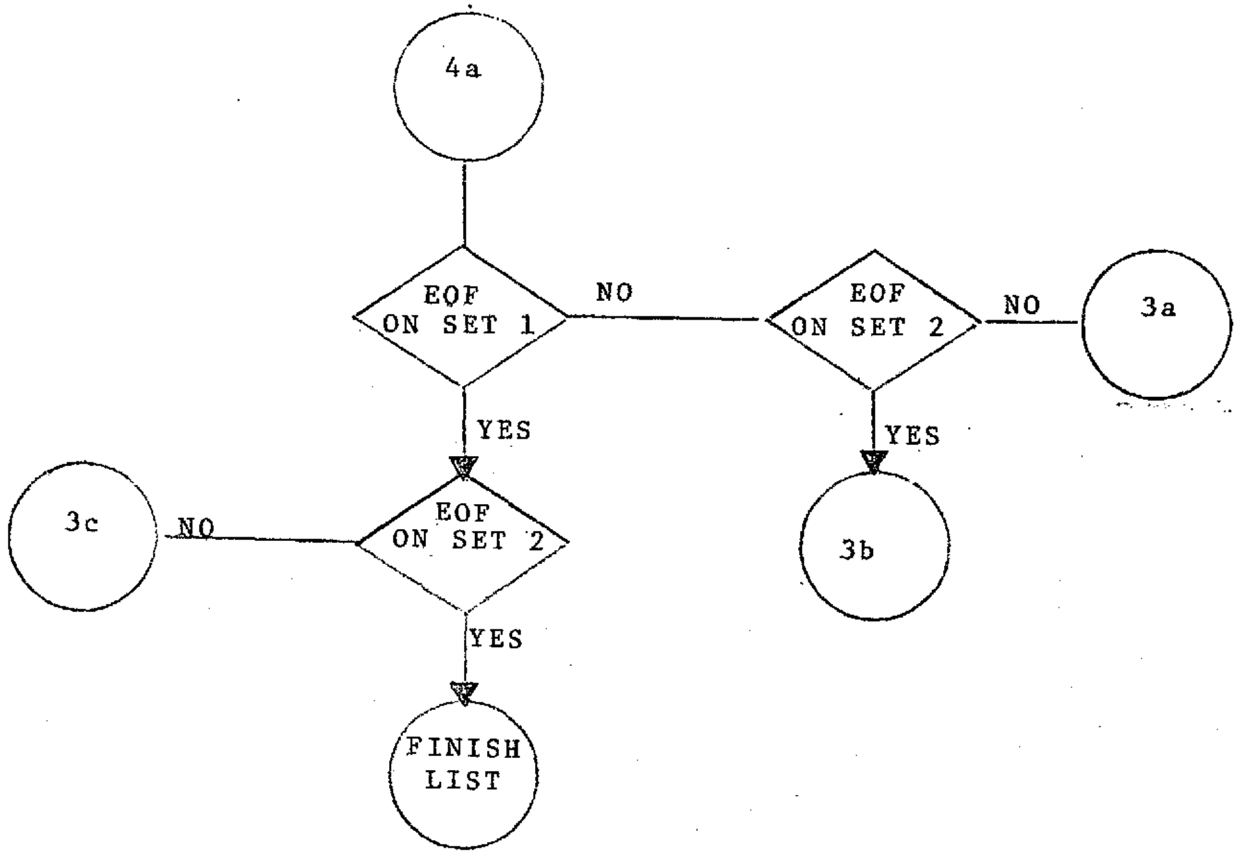
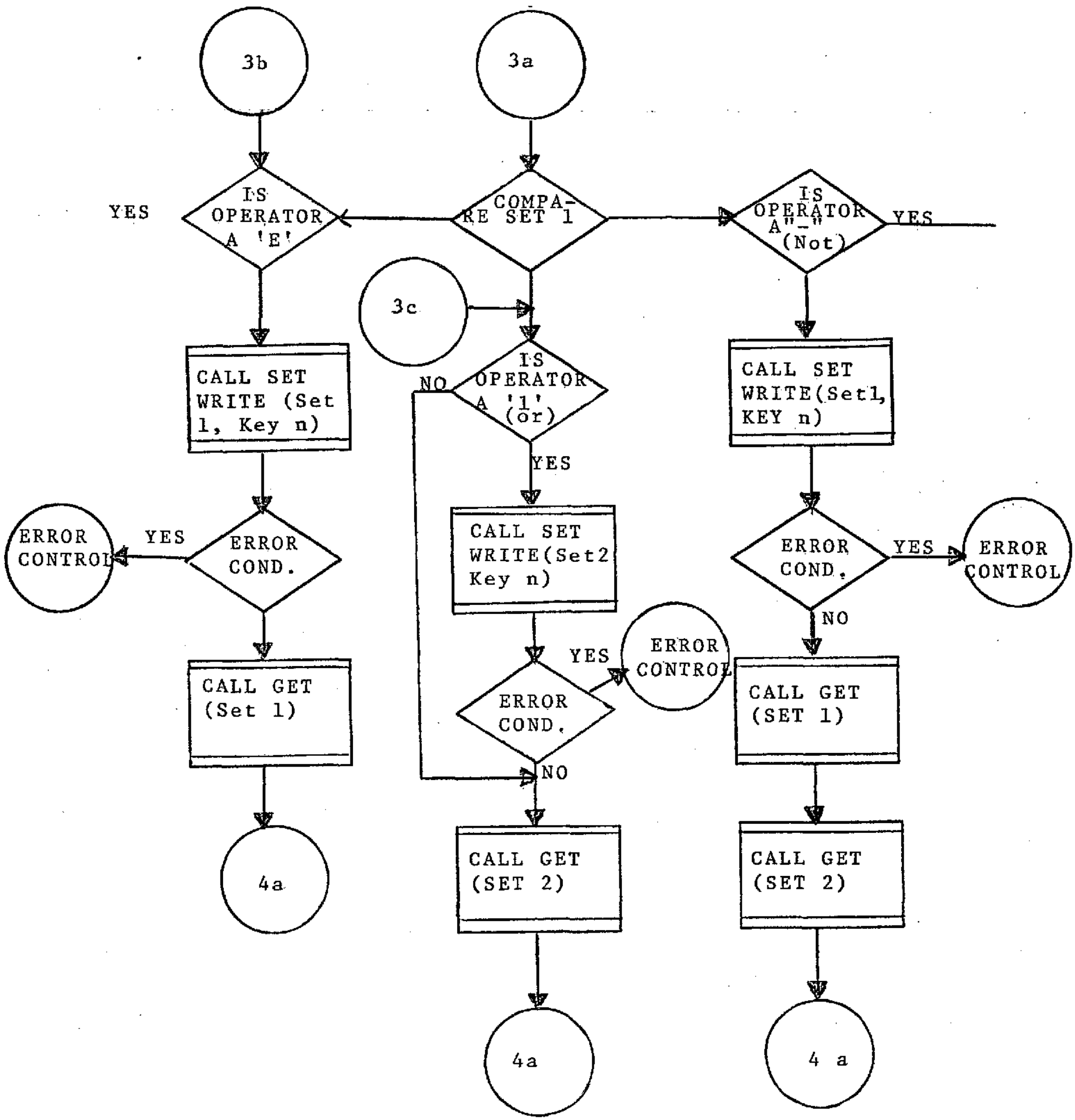


Figure 1g. LIST Entry (Page 3 of 4)


3b
3a
IS OPERATOR A 'E'
YES
COMPARE SET 1
IS OPERATOR A "-" (Not)
YES
3c
CALL SET WRITE (Set 1, Key n)
IS OPERATOR A '1' (or)
NO
YES
CALL SET WRITE(Set1, KEY n)
ERROR CONTROL
YES
ERROR COND.
CALL SET WRITE(Set2 Key n)
ERROR COND.
YES
ERROR CONTROL
NO
CALL GET (Set 1)
YES
ERROR CONTROL
ERROR COND.
CALL GET (SET 1)
NO
CALL GET (SET 2)
CALL GET (SET 2)
4a
4a
4 a

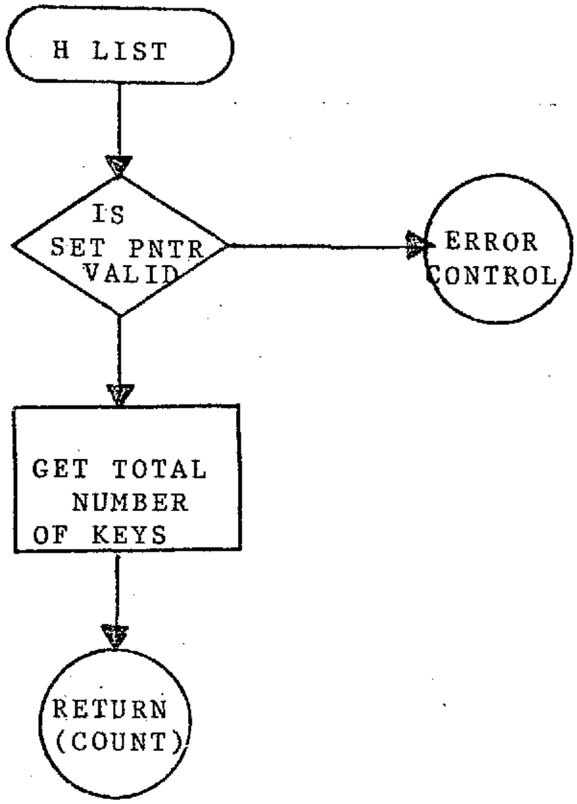


Figure 1h. #LIST Entry

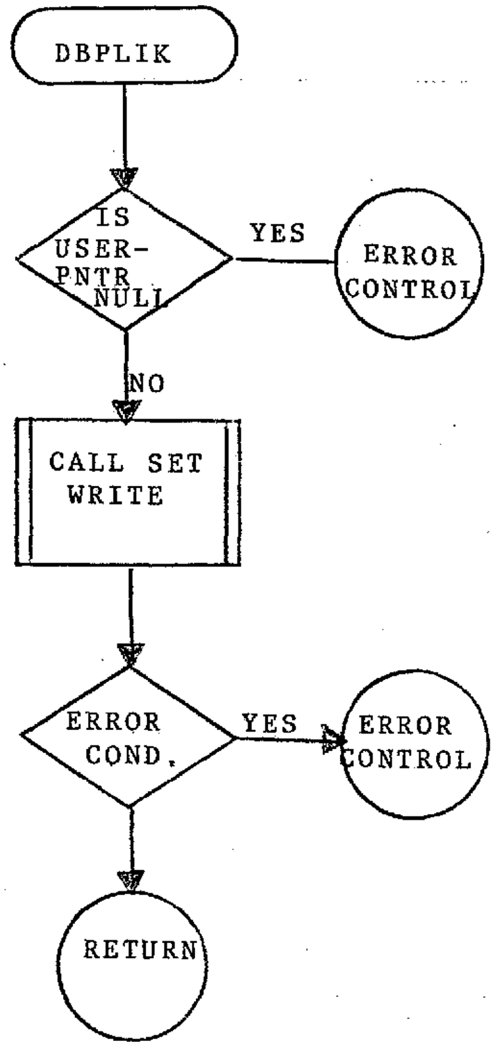


Figure 1i. DBPLIK Entry

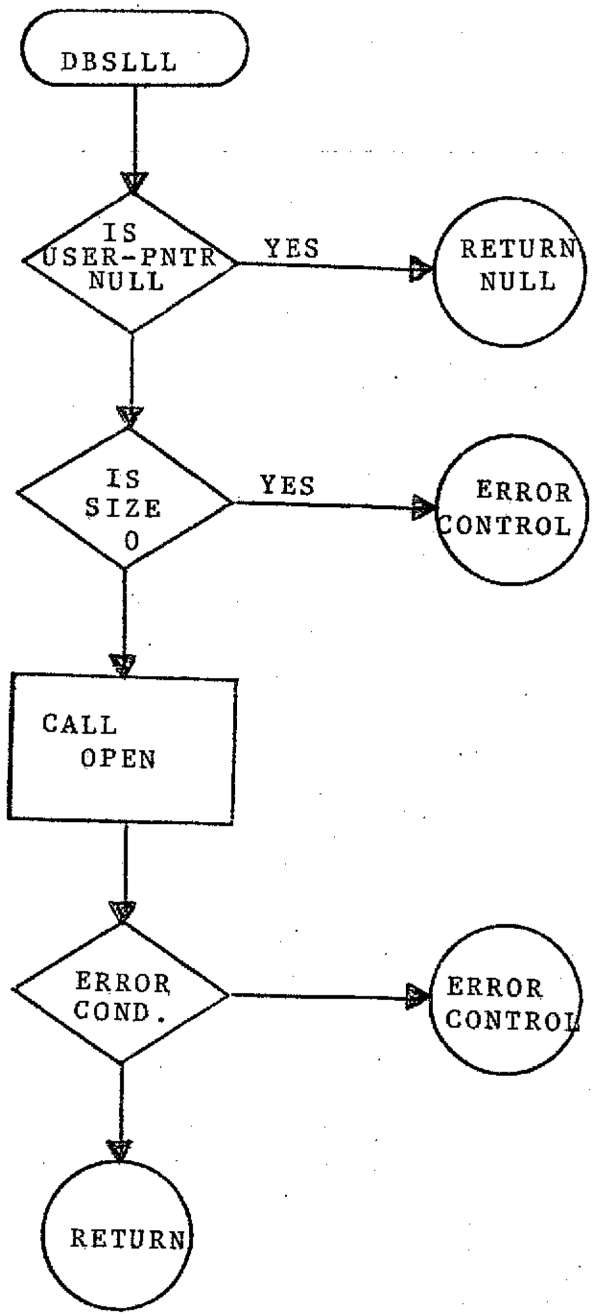


Figure 1u. DBSLLL Entry

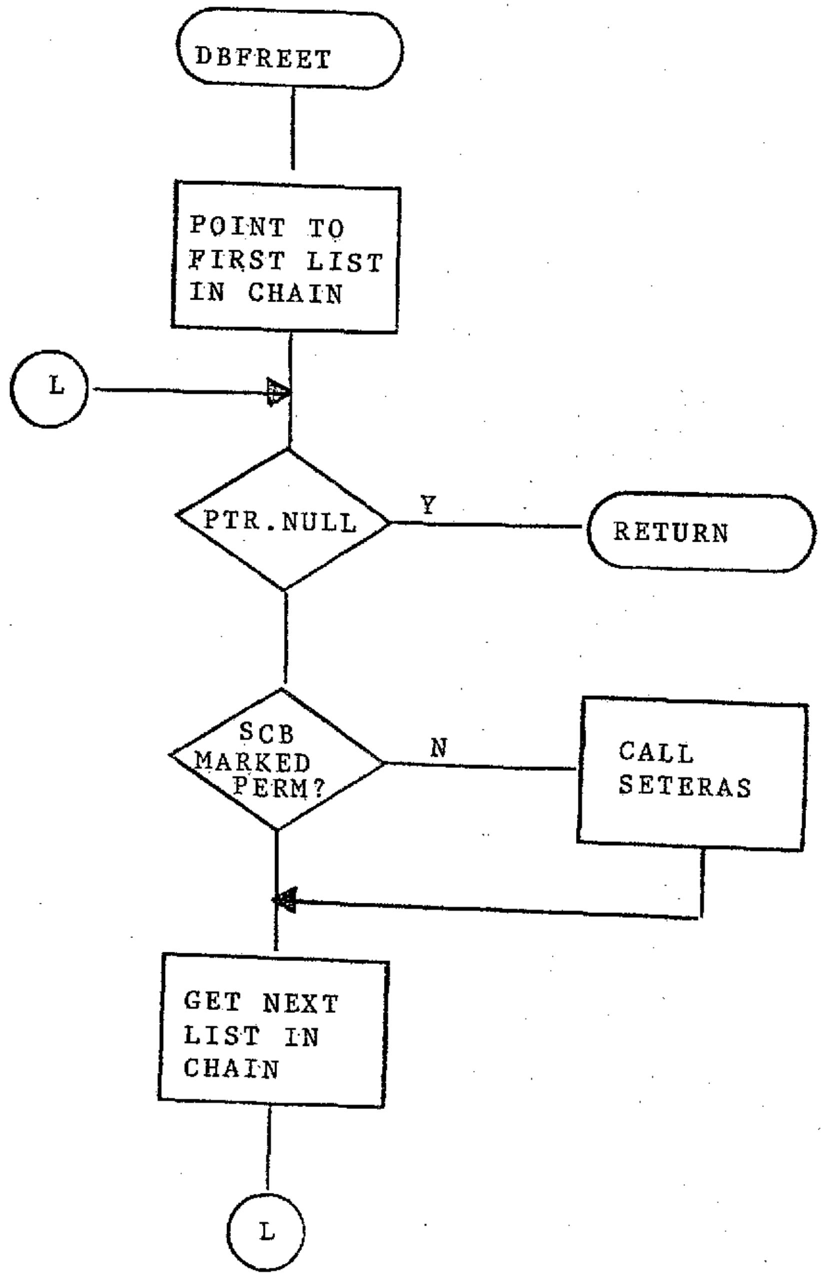


Figure 1k. DBFREET Entry

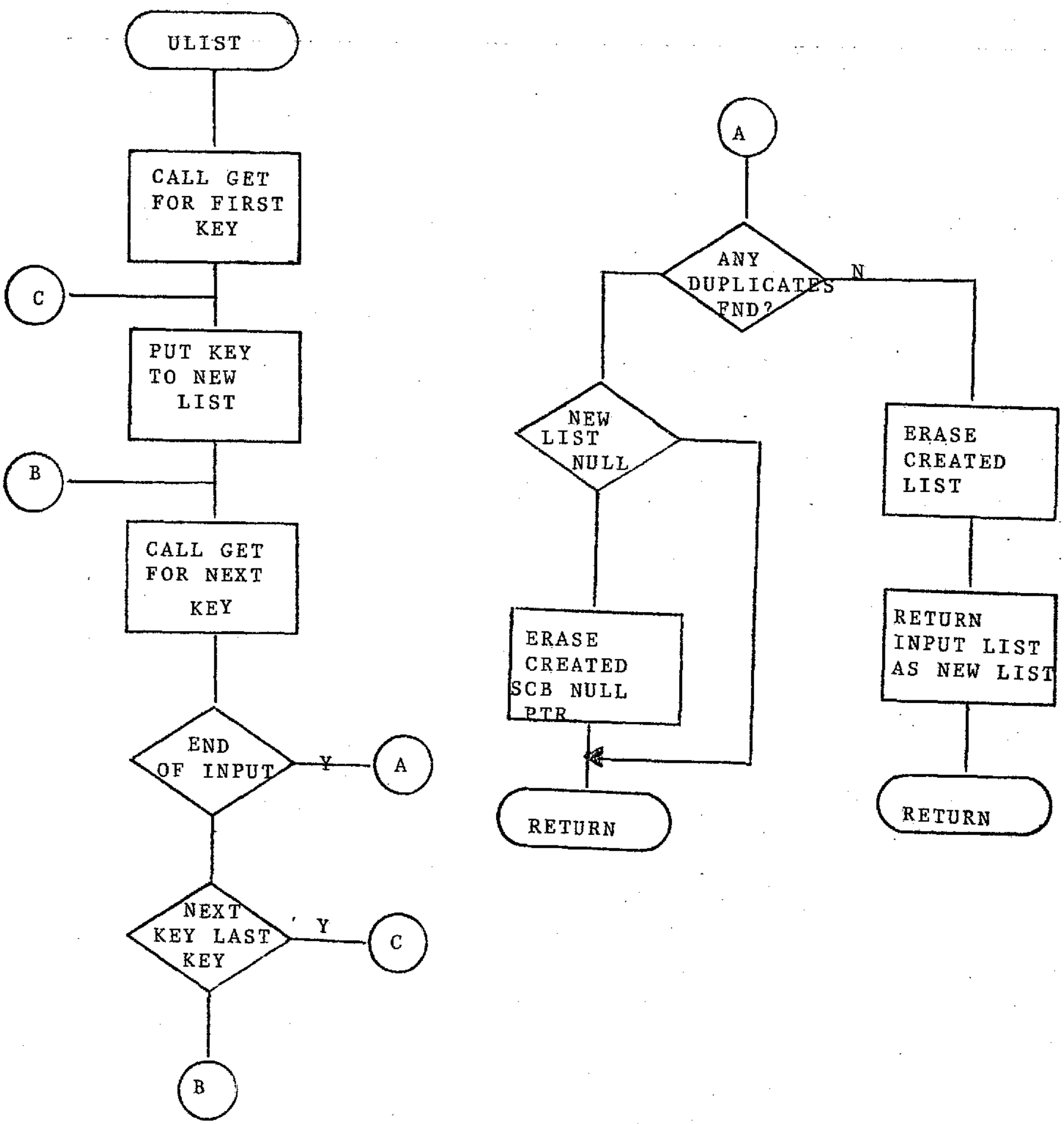


Figure 11. ULIST Entry

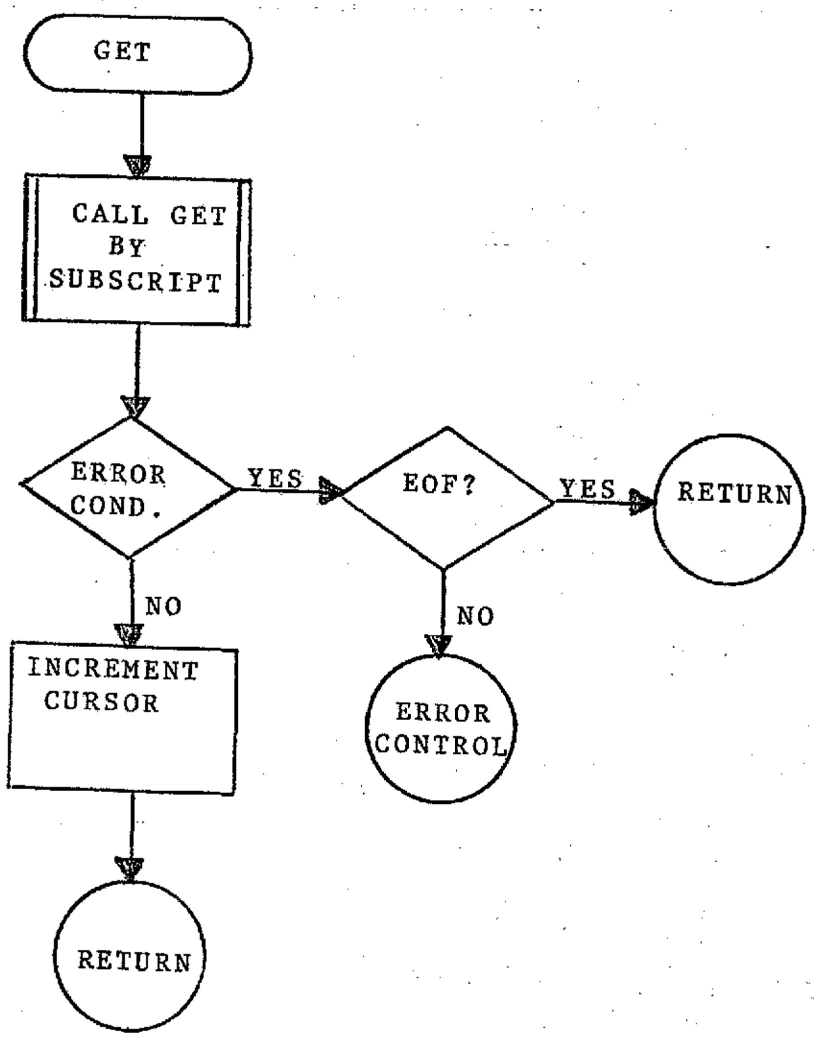


Figure 1m. GET Internal Routine

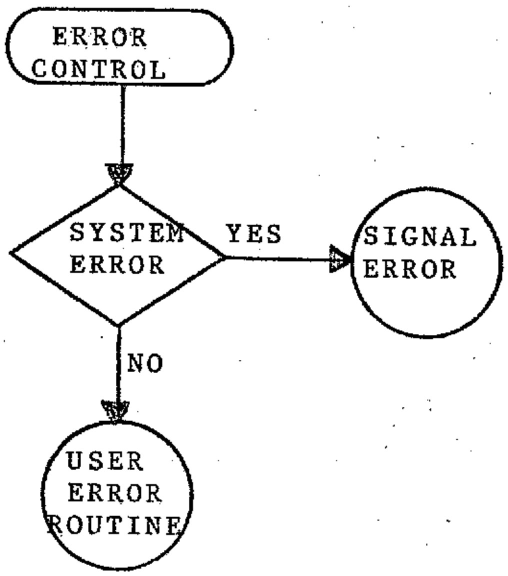


Figure 1n. ERROR CONTROL Internal Routine

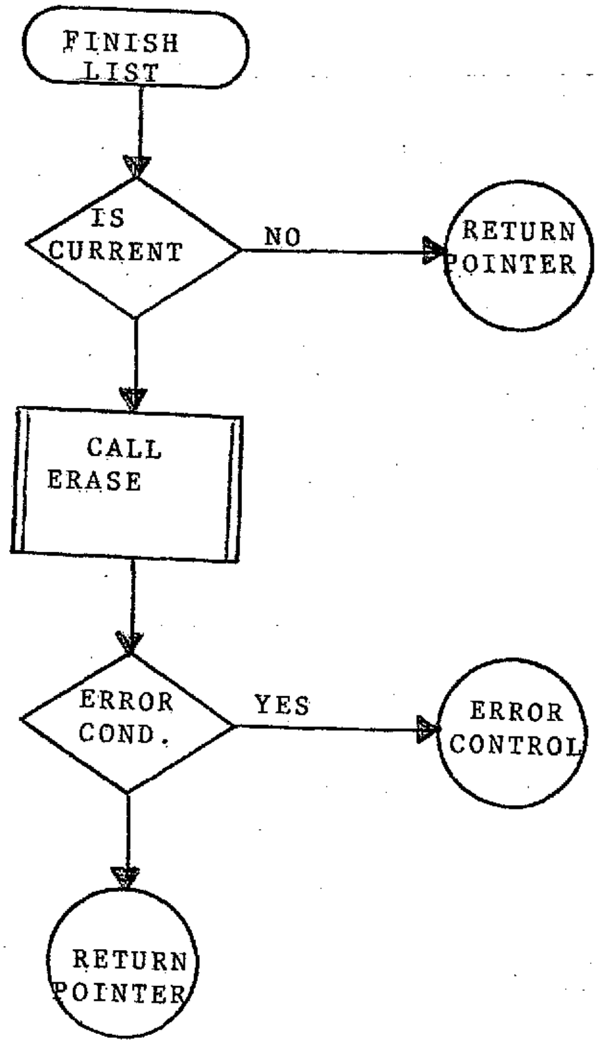


Figure 10. FINISH LIST Internal Routine

TOPIC B.6 - DATA BASE EXECUTIVE PARENT - CHILDREN FUNCTIONS

A. MODULE NAME

Data Base Executive Parent and Children List Functions.
Program-ID : NCCLIST
Module-ID : CCLIST
Entry points - UPLIST,CPLIST

B. ANALYST

William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

CCLIST builds a list of children (or parent) keys from a list of parent (or children) keys. Since data base input is needed to build these lists, this program could have been a part of DBPAC. However, (1) DBPAC is already large, (2) these functions are logically "above" those in DBPAC, in fact they use DBPAC and (3) these functions are infrequently used, i.e. only for certain SELECT operations when subfiles are involved. Consequently this is a separate module.

Mainline PL/I programs use the CCLIST services by function reference in a PL/I expression.

D. DATA REQUIREMENTS

See the DBPL/I Language Extension User's Guide, Section 8. Topic B.2.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

The routines all receive a MFCB (Mainline File Control Block) as their first parameter. They utilize entry points in the DBLIST or DBOSET modules to perform list or set operations. CCLIST's second parameter, a subfile control field name, is posted in MFCB.ONFIELD for DBPAC. The routines all receive a list pointer parameter. If it has the NULL value, they return a NULL list pointer immediately.

Then READ FILE LIST KEY (0); is done to reset the READ cursor of the input list and the list's key field name is compared with the anchor key field name in the core descriptor tables. For CCLIST they should be equal (the input list should be an anchor key list); if unequal, the input list pointer is returned immediately. For CPLIST and UPLIST they should differ (the input list should be a subfile key list); if equal, the input list pointer is returned immediately.

The #LIST function is invoked on the input list to obtain the count to govern further processing.

For the CCLIST function READ FILE LIST NOLOCK is done iteratively to process all (parent) records in the input set. From each one a GET FILE SUBFILE LIST SET is done using the second parameter for the subfile name. This returns a list pointer to a temporary set consisting of the subfile control field. If it is null, control loops back to the next READ and GET. If it is the first non-null control field encountered, it is made the basis for the output list. If it is a subsequent non-null control field it must be merged with the previous output. Control last segment. Otherwise (rarely) the OR LIST function of the RDBLIST module must be invoked to loops back until all the READ and GET's have been processed. The output list pointer is returned.

For the UPLIST and CPLIST functions a switch is set indicating whether duplicate keys are to be dropped after the parent list has been built by code common to both entry points. READ FILE LIST NOLOCK is done iternatively to process all subrecords in the input set. From each one the internal parent key value is extracted and posted to the output list.

If the output list has only one key, its pointer is returned for either UPLIST or CPLIST. Otherwise the output list is assumed to be in an ascending collating sequence. For the CPLIST function the output list pointer is returned at this point.

For the UPLIST function the previous key added to the output list is saved. New keys are compared to the saved key; if it is unequal to it, the new key is added to the output list and becomes the next saved key. The output list pointer is returned.

F. CODING SPECIFICATIONS

1. Source Language

PL/I

The SCB and LISTERR declarations are included from the SOURCE.LISRMAC dataset. Declarations for MFCB, DESC, DESC_FLD and FCB structures have been taken from the source for DBPAC.

No assembler routines are used.

2. Suggestions and Techniques

Not Applicable

TOPIC B.7 - EXECUTIVE ASSEMBLER ROUTINES

A. MODULE NAME

Program-ID: NDBRTNS
Module-ID: DBRTNS

B. ANALYST

Connie D. Becker
Neoterics, Inc.

C. MODULE FUNCTION

This program is divided into many routines and entry points each with a unique function. Refer to the individual entry point processing requirements in section E.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1.

2. Narrative

a. RETRNPT Entry

This routine is used by the Data Base Executive error routine. It posts the double word in the MFCB so that the user (of DBPAC) can return to the next sequential instruction in his program after the instruction in his program after the occurrence of an error. The first word is the invocation count; the second word is the address.

b. ASMMODE Entry

This routine is used to determine if the maintenance task is running in a batch mode. It returns a 'C' if running conversationally; or it returns a 'B' if not. The current implementation always returns a 'C'.

c. DBUCHEK Entry

This routine is used to validate the construction of an external name. The rules used are:

1. the name must begin with an alphabetic character (including #, $, @),

2. the name must be eight characters or less,

3. the second and subsequent characters of the name must be alphanumeric (including #, $, @, ?).

The parameters passed are the name and the name length (in the event that the user wishes to restrict it to less than eight). If the name is invalid, the length parameter will be set to one, as an error indicator, otherwise it will be set to zero.

d. MTT Interface Entries

ASMXTR, ASMPASS, ASMMUST: These entry points simply transfer control to the MTT monitor to maintain linkage conventions.

e. ASMID Entry

This entry is called to provide the system with the current installation ID. Current implementation returns an ID of 'NASIS***' to all calling programs.

F. CODING SPECIFICATIONS

1. Source Language

DBRTNS is written in IBM 360/OS Assembler Language

2. Suggestions and Techniques

Special attention is paid to the linkage conventions of the current PL/I compiler.

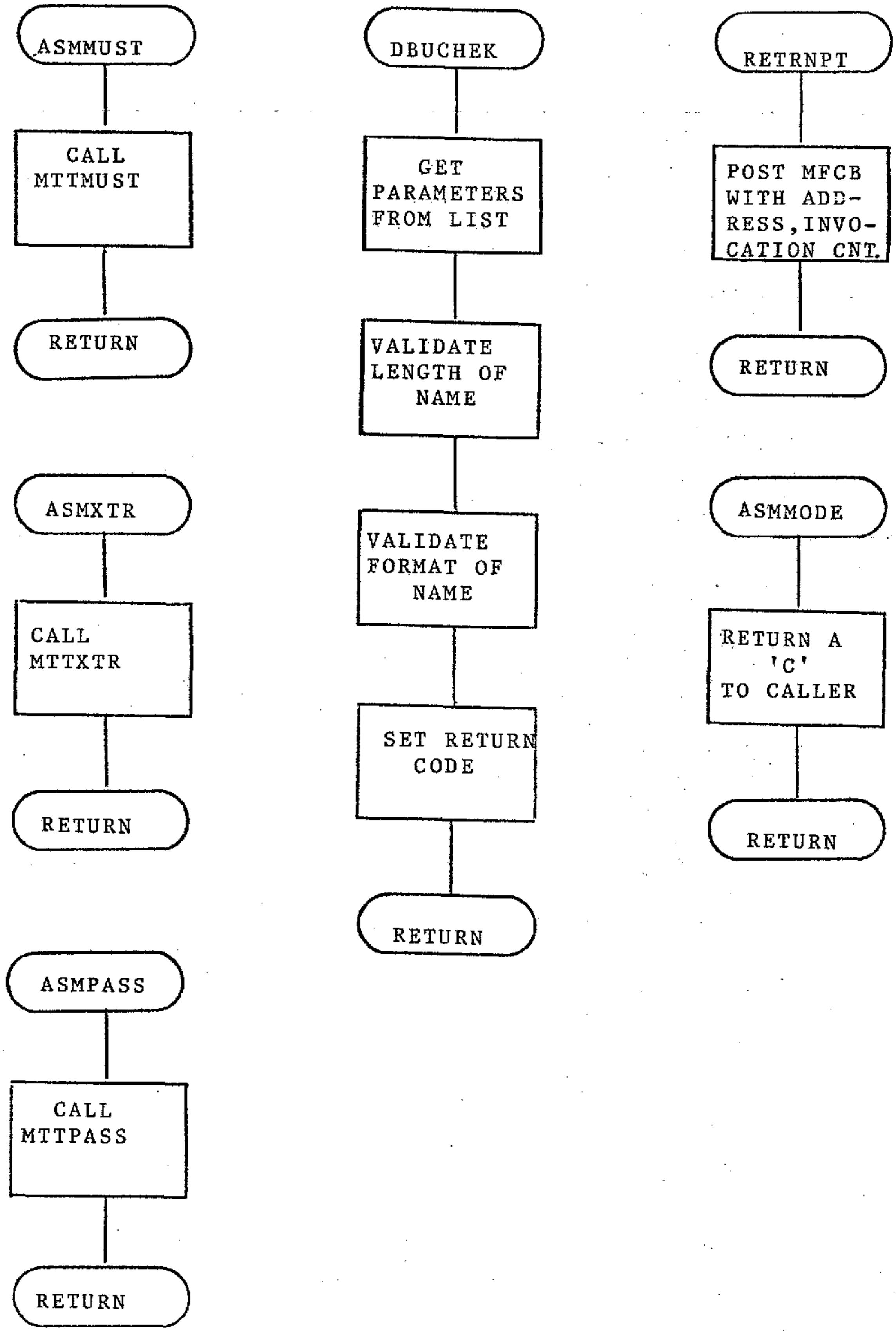


Figure 1, TOP LEVEL FLOWCHART

TOPIC B.8 - SET MANAGEMENT

A. MODULE NAME

Program-ID: NDBOSET
Module-ID: DBOSET

B. ANALYST

O. K. Hearne
G. P. Oswald
Neoterics, Inc.

C. MODULE FUNCTION

NDBOSET will provide the OS system with the means of storing and retrieving SET's. These SET's will be maintained as a single data set for all users. The module will provide the capability of dynamically allocating and deallocating both external storage areas and internal buffers. The life span of a SET is currently the duration of a session or from LOGON to LOGOFF (which ever is shorter).

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1.

2. Input Data Sets

   a. Parameter Cards

      Using the normal OS procedure for passing parameters in the EXEC card, the following information will be available at load time:

      1. Number of records in an allocation group.

      2. Allocation algorithm variables (proposed).

      3. Number of set I/O buffers to be allocated.

   b. Punched Card Input Files

      None

   c. Input Files (Proposed)

      The standard NASIS index files may, at times, be input to the routine when such a

file has been designated as a set. Keys may be retrieved via normal facilities.

d. On-Line Terminal Entries

None

3. Output Data Sets

a. Output Files

The NDBOSET module is primarily concerned with a single large BDAM data set. All sets of keys are allocated as subfiles; example, SET number is subfile name within the primary data set. Records are accessed using relative record numbers. Bit tables are used to indicate which records are allocated to a particular set. The bit tables allow fast deallocation of records comprising a set.

The primary BDAM file is defined using standard JCL cards. Record size, file size and palcement can be specified for maximum optimization for an installation. The parameterized information will be used to preformat the allotted storage areas and control the construction of the SET manager's system tables. The above preliminaries will be a function of the system load routine, thereby reducing the SET manager's resident housekeeping requirements.

b. File Attributes

1. Organization - BDAM

2. Recording Medium - DASD

3. Name - SETPLEX

c. Record Attributes

1. Key Identifier - None

2. Length - As Specified in JCL

3. Mode - FIXED

4. Blocking Factor - 1

d. Field Attributes

1. Length - Same as Key Length

2. Mode - Fixed at Key Length

3. Data Type - Alphanumeric

4. Position - As many fields (keys) as will fit in record.

e. Other Identifying Information

The BDAM logical record size and block size must be equal. NDBOSET itself provides for blocking and deblocking of the keys into the BDAM record.

4. Reference Tables

The SET manager's system tables are installation variables and, therefore, the construction and initialization of the system tables will be a function of the system loader. These tables are likewise expected to be protected from extraneous manipulations.

a. Master Bit Table (MABTAB) Global System Table

This table will be comprised of four parts. They are as follows:

1. Master Table Header Block

The header will contain static information defining the characteristics for the SET data set (SETPLEX).

2. Master Segment Control Block

This section of the table requires one entry for each segment (device) allotted to the SET data set. It will contain both static and variable information.

3. Master Bit Table Block

This part of the table will be formated so as each group (of records) contained in the SET data set has a corresponding bit in the table whose integrity is maintained by the SET Manager's allocate and deallocate functions.

4. The Section Lock Table

    This table is the same size as the Master Bit Table. It contains a one byte entry for each section (8 bits equal 1 section). The table is used by the SET Manager to lock a particular section of the Master Bit Table. In this manner the SET manager is able to maintain the integrity of a given section during task allocation.

There are several reasons for this type of control on the SET data set. They are as follows:

1. The use of allocation by group allows the record size tc ce independent of the amount of storage taken in each allocation.

2. Faster allocation of records since a current status of all records available in the SET data set is possible by a simple interrogation of the bit table.

3. Deallocation of a SET (ERASE) requires resetting of all bits associated with the SET. No record manipulation is necessary.

4. Core storage is minimized since each bit position represents a unit of records (group).

b. Master Bit Table Block Characteristics

There are three levels of the Master Bit Table Block.

1. Segment. A segment is defined as a predetermined area residing on a physical device which has been mapped according to the system's specifications as part of the SET data set. There are as many segments as there are physical devices assigned to the SET data set.

2. Section. A section is an internal unit of organization within a given segment. The number of sections within a segment is a function of the record size and the number of records per group as defined for a system. The section is represented by eight bits (byte) in the Master Bit

Table Block. The use of byte is for programmatical ease.

3. Group. A group is the smallest entity represented in the Master Bit Table. Each bit represents a group of records. The number of records per group is parameterized and, thus, allows for individual system optimization.

c. Buffer Pool

The buffer pool will be a predetermined size depending upon record size of the installation. This will be built at system initialization time and contain as many buffers as the system will allow. The format of pool allows a task to step through the buffers to find one which is not allocated. The buffer will be marked as locked and the current available buffer count will be decremented.

d. Section Lock Table

The Section Lock Table is comprised of a one byte entry for every section within the SET data set.

Allocation of a group(s) is attempted within one section. If that section cannot accommodate the number of groups requested, the next section within a segment is tested. Once a section satisfies the allocation requirement, it must be marked so that a manipulation of the group bits can occur. The Section Lock Table will reflect those sections currently being manipulated by the allocator and, thus, negate the possibility of multiple allocations of a single group.

FF indicates this section is currently being manipulated by the allocator.

e. User Set Table (USETAB) Local Task Table (One Per Set)

The User Set Table will be comprised of two parts. They are as follows:

1. Set Control Block

The Set Control Block will contain the current information regarding a user's SET.

2. User Bit Table Block

The bit table will retain the allocation characteristics of an individual SET. Each UBTB will reflect the section number and eight allccation bits within that section.

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

See Figures 2.1, ff.

a. READ/WRITE LOGIC

All read operations require a logical record # a buffer and a key position within the buffer. Each entry point will be responsible for setting a current key subscript in the appropriate SCB. In this manner a common routine can be utilized to decide whether the key requested resides within the realm of the current group; ie, 1 group covers X records and Y keys four records.

1. The write routine will set the subscript to the previous subscript + 1

2. Sequential read forward will set the subscript to the previous subscript +1

3. Sequential read backward will set the subscript to the previous subscript +1

4. Subscripted read will set the subscript equal to the one requested.

Division of the subscript by the number of keys per record equals the relative record number and the key position within the record if the relative record number is equal to the previous record number, the requested key location is within the current record. Division of the relative record number by the number of records per group will develop a remainder equal to the relative pcsition of the logical record number within a group.

Example:

Number records per group	=	4
Number keys per record	=	20

```
          Previous subscript              = 107
          Previous logical record #-1     =  10
          Previous relative record &
             key position                    5, 7
          File layout

Log Rec 1: Rec 0, keys 0-19
           Rec 1, keys 20-39
           Rec 2, keys 40-59
           Rec 3, keys 60-79
Log Rec 9: Rec 4, keys 80-99
           Rec 5, keys 100-119
           Rec 6, keys 120-139
           Rec 7, keys 140-159
```

Two groups have been allocated to this file. They control logical record numbers 1 thru 4, and 9 thru 12.

b. NDBOSET will provide the following internal routines which will be invoked by external calls.

1. Set Write

2. Set Read

3. Open Set Write (Currently Implied Entry)

4. Open Set Read (Currently Implied Entry)

5. Close (Currently Implied Entry)

6. Erase Set (Currently Implied Entry)

These routines will comprise the normal processing path within the module.

Each of the above routines will require an internal call to the various managers within the module. The following are the manager entry points:

1. Master Bit Table Manager

   a. Allocation

   b. Deallocation

2. Set Control

3. User Bit Table Manager

4. Core Request/Release Manager

5. Read/Write Key Manager

6. Buffer Manager

7. Error Control Manager

8. Subscript Function

9. Resequence Function

10. OR Function

Before the system (session) is operational, there are two phases of initialization which must be accomplished:

1. Preformatting of SET data set records.

2. Table allocation and initialization.

Step one of the above may be an off-line program unless reorganization of SET data set is required at system load time.

Step two is required every time the system is initialized. Step two is a parameterized housekeeping function which need only be called once during a session. It is, therefore, a completion of its function. See system load Primary initialization of SET data set.

c. Routines Invoked by External Calls:

There are two basic entry points to the module. They are: SET WRITE and SET READ. These external requests will require the remaining functions of the SET Manager. However, further analysis of the system has shown a desirability to provide additional external requests in order to efficiently maintain system and core resources. Therefore, an additional four functions will provide for external calls. They are as follows:

1. SET WRITE

   The functions are:

   a. Verify existence of the SET number.

   b. Verify SET is in write mode.

c. Non-existent SET implies OPEN of new SET.

d. WRITE key to SET.

2. SET READ

a. Verify existence of the SET number.

b. Verify SET is in read mode.

c. Existing SET in write mode implies CLOSE and OPEN of SET.

d. Read key from SET.

3. OPEN SET WRITE

a. Call SET control to create new SET.

4. OPEN SET READ

a. Verify existence of SET.

b. Verify SET closed.

c. SET OPENed for WRITE, must be closed to force out last record and reOPEN.
d. Deallocate unused groups.

5. CLOSE

a. Verify existence of SET.

b. CLOSE SET

1. In WRITE mode, must force write.

2. CLOSE READ SET.

c. Call SET control for updating SET Control Table.

d. Call Buffer Manager for release of I/O Buffers.

6. ERASE SET

a. Verify existence of SET.

b. Call SET control for updating SET Control Table.

c. Call Buffer Manager for release of I/O Buffers.

d. Call Deallocation for release of allocated groups.

e. Call SET control for release of SET Control Table.

d. Functions Invoked by Internal Calls:

The purpose of these functions is to maintain the integrity of the user SETs. Whenever possible, these functions will perform as part of the user task.

1. Master Bit Table Manager (GETRCD)

    a. Allocation

        1. Determine segments for attempted allccation.

        2. Find section within segment to accommodate allocation.

        3. Lock section during bit manipulation (allocation).

        4. Call User Bit Table Manager.

    b. Deallocaticn (ERASE)

        1. Call SET Control for updating of SET Control Table.

        2. Obtain each entry within User Bit Table Block.

        3. Mark corresponding section/group bits as unallccated.

        4. Call Core Manager to release User Bit Table entries and Set Contrcl Block.

2. Set Control

    a. Call Core Request/Release Manager for create/destroy of SET Control Table.

    b. Update SET Control Table variables.

3. User Bit Table Manager

    a. Insert newly allocated section in User Bit Table.

    b. Determine if another entry is possible.

        1. Call Core Request/Release Manager.

        2. Initialize new User Bit Table.

        3. Link User Bit Table entry.

4. Core Request/Release Manager

    a. Parameterized request of memory.

    b. Parameterized release of memory.

    c. Indication of insufficient memory requires the task to wait.

5. Read/Write Key Manager

    a. Write

        1. Insert Key in current buffer.

        2. Write buffer, if full.

        3. Call Buffer Manager.

        4. Wait User Task.

    b. Read

        1. Obtain Key from current buffer, if possible.

        2. Call Buffer Manager, if no buffer.

        3. Initiate Read.

        4. Wait User Task.

6. Buffer Manager

    a. Obtain or relinquish buffer from buffer pool.

    b. Call Core Manager.

1. If no buffers available.

2. If buffer returned causes previously requested memory to be free.

7. Error Control Manager

   a. Facilitate debugging aids.

   b. Traps system errors.

8. Subscript Function

   The Subscript function will allow a user to request a key from an existing SET by subscript. The user has three (3) options for accessing a record by subscript.

   a. Positive Subscript - Direct displacement to key from relative key zero.

   b. Zero Subscript - Resets current key pointer to beginning of SET.

   c. Negative Subscript - Sets current key position to last key in SET or to current key position - 1.

9. Resequence Function

   As each write key request is processed, the write manager will test for a possible key out of sequence. This is possible within children/parent list operations.

   The resequence function will first attempt to resequence the keys within the current user buffer. If it is determined that the out-of-sequence key must affect a previously written record, a OR set will be opened and a key that fits this criteria will be kept in sequence in the OR set.

   The 'OR' set can fill a buffer; if this happens, the user set will be pseudo closed and the OR set and the OR function of the set manager will be involved.

10. OR Function

    The OR function will be used to generate a new set by ORing the user's set with the resequence 'OR' set. An entry will be made

into the LIST functions which facilitate this requirement. Upon completion of the new combined set, the SET Manager will perform the following operations:

a. Deallocate user's set.

b. Overlay user's SCB with the new combined set's SCB.

c. Release new combined set's SCB.

d. Release 'OR' set's SCB. (No deallocation necessary; 'OR' set maintained as single internal buffer.)

Control will return to the Write Manager for 'write key completion' processing.

e. Requirements Outside of NIBOSET:

1. Before System Load

   a. Primary BDAM file initialization, preformatting and description.

2. System Initialization

   a. Build tables.

   b. Get number of records/group.

   c. Open BDAM data set.

3. Monitor

   a. WAIT with single ECB address.

4. I/O Manager

   a. Write BDAM record using relative record number.

   b. Read BDAM record using relative record number.

5. System Shutdown

   a. Close BDAM data set.

6. Logoff Processor and User Abend

   a. Call to NIBOSET to flush and erase

user's sets.

7. NASIS Director

   On return from command processing modules, a call to NDBOSET is necessary to release buffers still allocated and other clean-up.

8. Command Modules

   CLOSE or FREE LIST can be used to close a set and deallocate buffer.

9. DBPAC

   a. Provision to allow GET INDEX ELEMENT forward, backward and by subscript.

   b. Separate READ and GET pointers which are not maintained by the SET Manager. DBPAC must use read by subscript to maintain position.

f. I/O Manager Linkage:

   Entry Point: SETIO

   R1 - Points to parameter list in core.

   1. Parameters

      a. DECB address (FW)

      b. Relative record number (FW)

   2. DECB Contains:

      a. Buffer Address

      b. Read or Write Flag

F. CODING SPECIFICATIONS

1. Source Language

   Assembly Language is used for efficiency.

2. Suggestions and Techniques

   a. Needs on Set Manager

      1. A request to set the GET cursor (current key pointer) to the beginning of the

SET. (This call could be a pseudo OPEN READ.)

2. Subscripted read from a set.

3. Design macros for call to Set Manager.

4. OPEN Processing

   a. If OPEN SET READ is implemented, then OPEN could return an error indicator for involved OPEN, for OPEN of NULL file, and for SET OPENed successfully.

   b. A return of a null entry when a SET exists but no keys are present in the SET.

5. Set Manager must retain field name and conversion routine name.

6. Set Manager must check sequence of keys on all WRITE's. When a sequence error is detected, a second list must be made available in order to retain the keys to be inserted. Whenever necessary, the two

7. One section may cross a segment boundary.

8. Conditional GETMAIN within NASIS.

9. Address of user's USETAB chain in user's master table.

10. Clean-up exit in LOGOFF processor.

11. Relinquish control macro.

12. Set closing function SCLOSE.

13. All sets are closed by return to director.

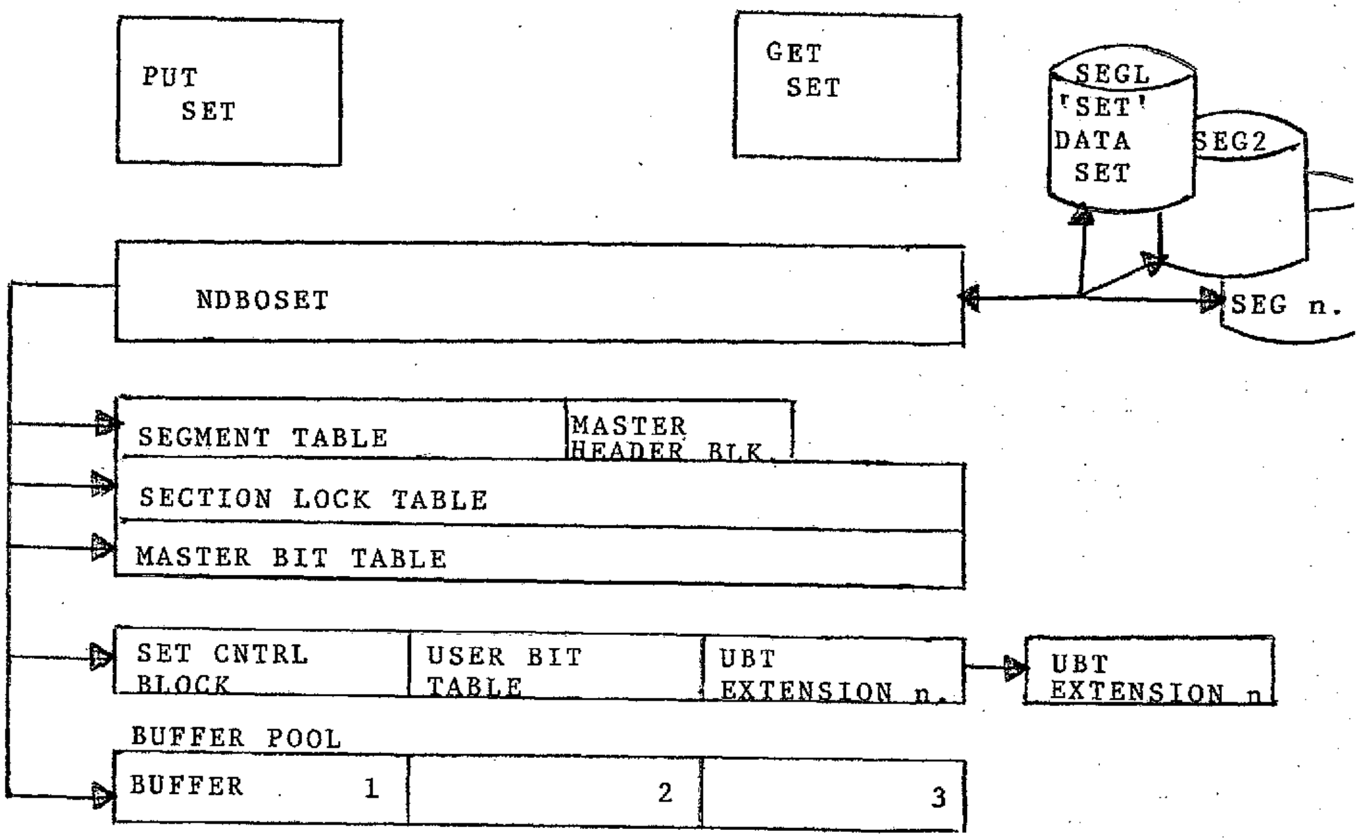


Figure 1. I/O BLOCK DIAGRAM

Offset		
0	MNXTAL ADDRESS OF NEXT SEGMENT TO BEGIN ALLOCATION	
4	MBGNMBT BEGINNING ADDRESS OF MBT	
8	MENDMBT ENDING ADDRESS OF MBT	
12	MISECLCK ADDRESS OF THE SECTION LOCK TABLE	
16	MCURBUF ADDRESS OF THE CURRENT BUFFER IN THE POOL	
20	MFRTBUF ADDRESS OF THE FIRST BUFFER TO POOL	
24	MNUMBUF NUMBER INTERNAL BUFFERS	MNUMSEG NUMBER OF SEGMENTS
28	MREGRP RECRODS PER GROUP	MBUFSIZ BUFFER SIZE
32	MRECSIZ RECORD SIZE	
36		
40		
44		
48		
52		
56		
60		
64		
68		
72		
76		
80		
84		

Table 1. MASTAB

0	1 BYTE PER SECTION - EACH BIT EQUAL 1 GROUP
4	
8	
12	
16	
20	
24	
28	
32	
36	
40	
44	
48	
52	
56	
60	
64	
68	
72	
76	
80	
84	

Table 2. MASTER BIT TABLE

MSTB

Offset	Contents
0	MSALOC MSSECNO
4	MSSEGAD ADDRESS OF NEXT SEGMENT TABLE ENTRY
8	
12	
16	
20	
24	
28	
32	
36	
40	
44	
48	
52	
56	
60	
64	
68	
72	
76	
80	
84	

Table 3. Master Segment Table

0	1 BYTE PER SECTION 'FF' INDICATES
4	ALLOCATION IN PROCESS
8	
12	
16	
20	
24	
28	
32	
36	
40	
44	
48	
52	
56	
60	
64	
68	
72	
76	
80	
84	

Table 4. Master Section Lock Table

Offset	Field	Description
0	BUFLOCK BUFNEXT	ADDR OF NEXT BUFFER
4	BUFFER	BUFFER EQUAL RECORD SIZE (FULL WORDS)
8		
12		
16		
20		
24		
28		
32		
36		
40		
44		
48		
52		
56		
60		
64		
68		
72		
76		
80		
84		

Table 5. BUFPOOL

0	SNXTSCB ADDRESS OF NEXT SCB	
4	SUBTDIS/SCURUBT UBT DISPLACEMENT/ADDR OF CURRENT UBT	
8	SCURBUF ADDR OF CURRENT BUFFER	
12	SCURKEY ADDR OF CURRENT KEY	
16	SFLD NAME	
20		
24	SCON RTN	
28		
32	SBGNREC BGN LOGICAL REC NUMBER	SENDREC ENDING LOGICAL REC #
36	SCURREC CUR LOGICAL REC NUMBER	SNUMREC NUMBER RECS WRITTEN
40	SKEVREC NUMBER KEYS PER REC	SKYLSTR NUMBER KEYS LAST REC
44	SKEYLNT KEY LENGTH	SCNTRL CONTROL BYTE
48	SNUMGRP NUMBER GRPS PER ALLOC	SRECCUR GET-RECORD CURSOR
52	SUBSCRPT	
56	SCBDECB	
60		
64		
68		
72		
76		
80		
84		

Table 6. SCB

Offset			
0	SPRVUBT		
4	SNXTUBT ADDRESS OF NEXT UBT EXTENSION		
8	UBTSECNO SECTION NUMBER	UBSTALCTL CONTROL BYTE	UBTALMSK ALLOC MASK
12			
16			
20	BCBPRKEY 255		
24			
28			
32			
36			
40			
44			
48			
52			
56			
60			
64			
68			
72			
76			
80			
84			

Table 7. UBT

BYTE	BIT POSITION	CONTENTS
0-1		Contains section number from which group(s) (Byte 3) were obtained.
2	0	Not Used Initialized to zero when UBTE created NUMBER OF GROUPS Allocated by this entry. Initialized to 1 when UBTE created
	0-3	CURRENT GROUP (Bit Position) Being filled (write) or ACCESSED (READ).
3		CONTAINS GROUP(s) ALLOCATED WITHIN THIS SECTION.

TABLE 8. USER BIT TABLE ENTRY

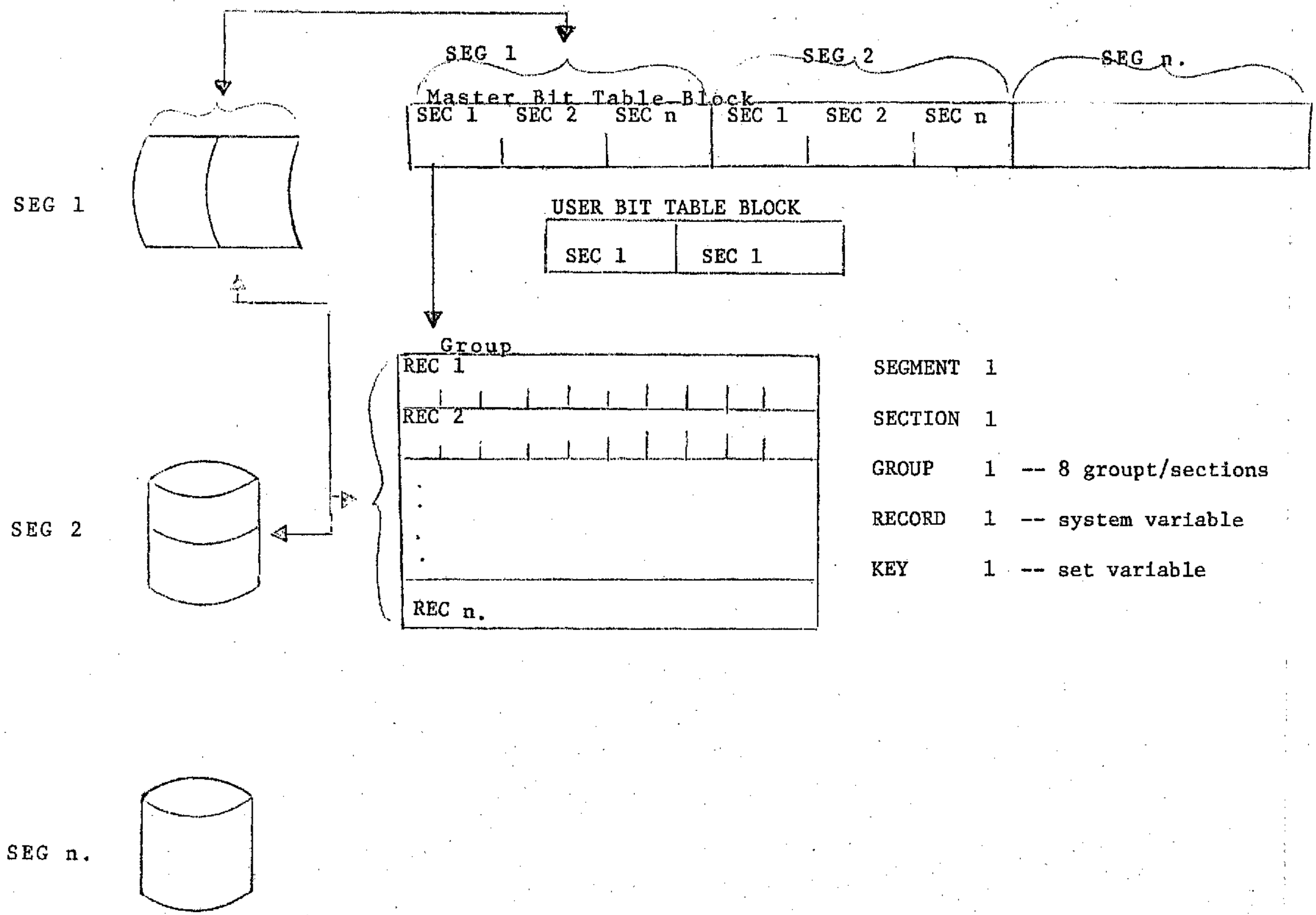


TABLE 10. FILE RELATION SHIPS

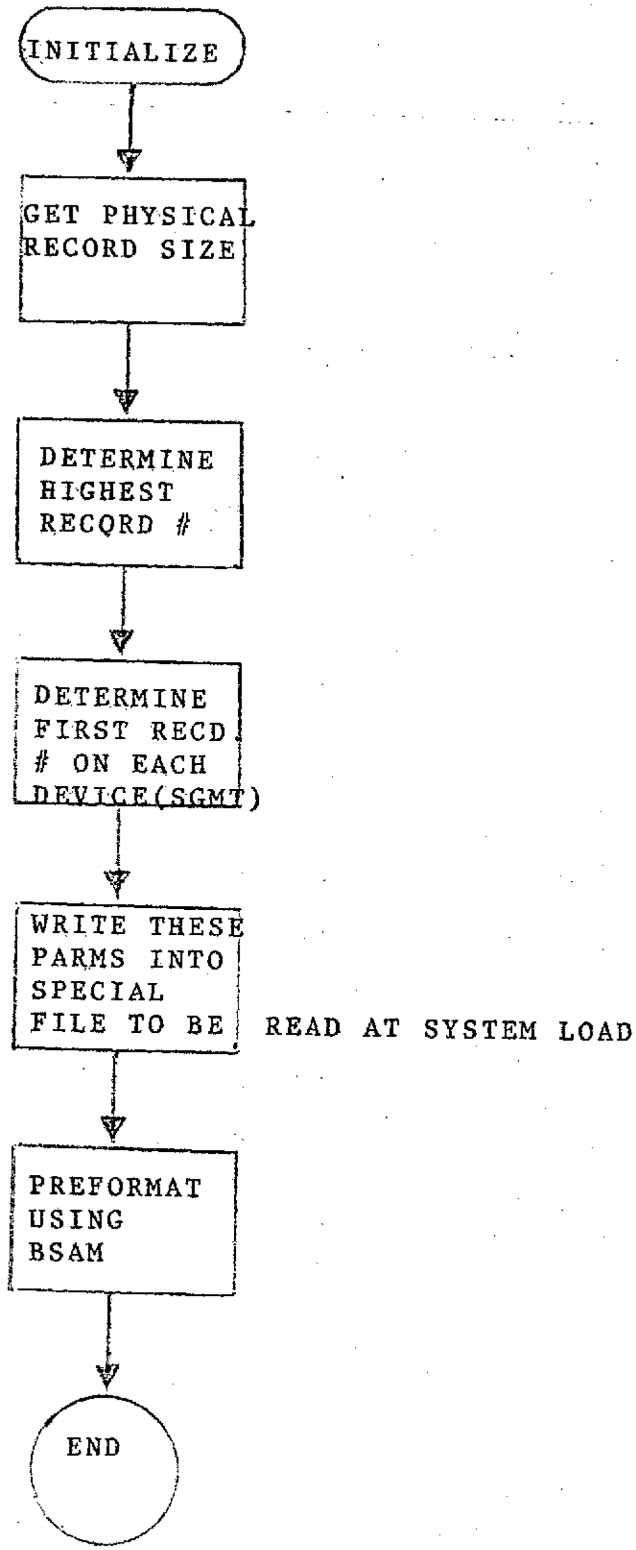


Figure 2.1 FILE INITIALIZATION
(Must be done before system load)

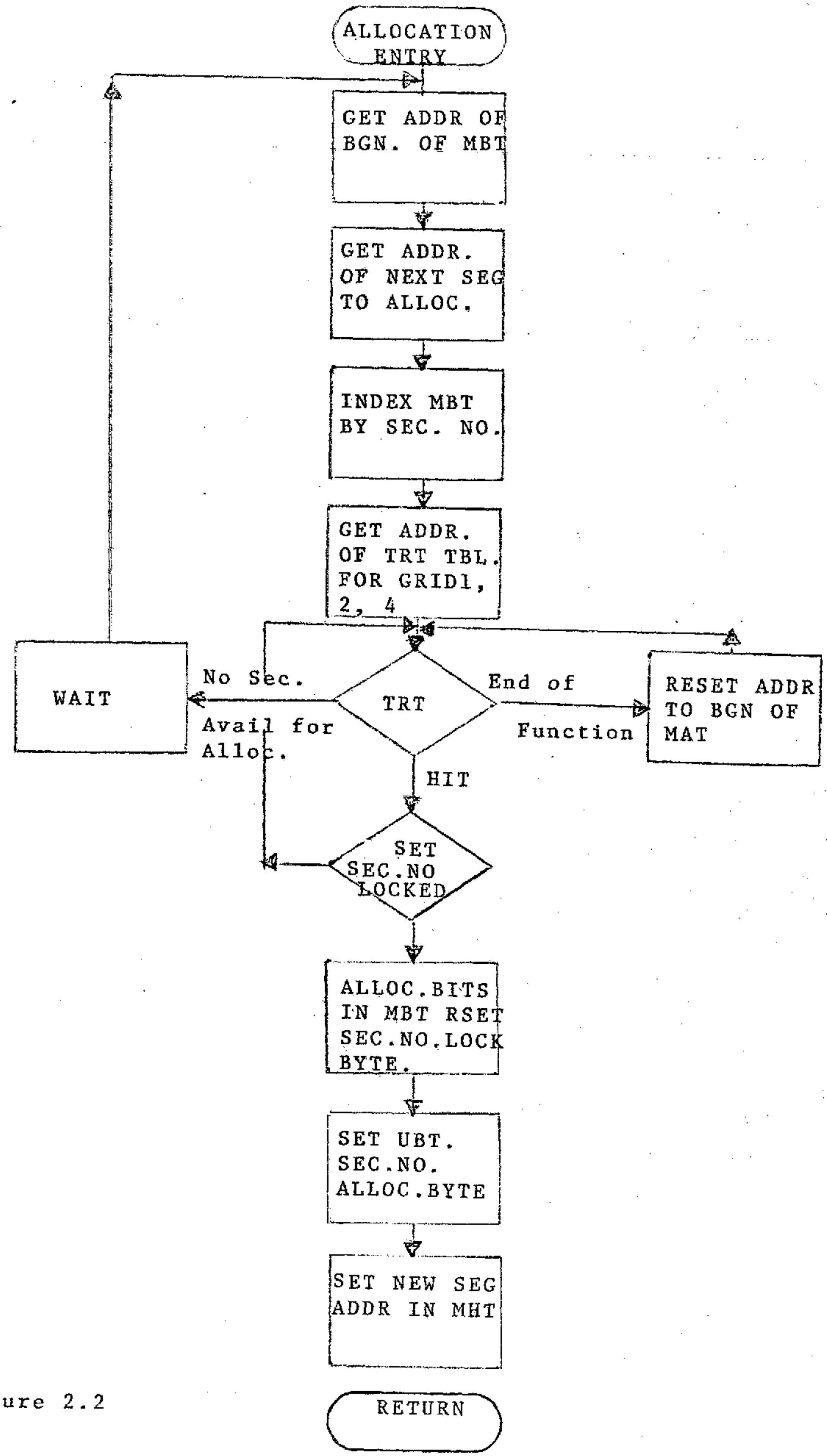
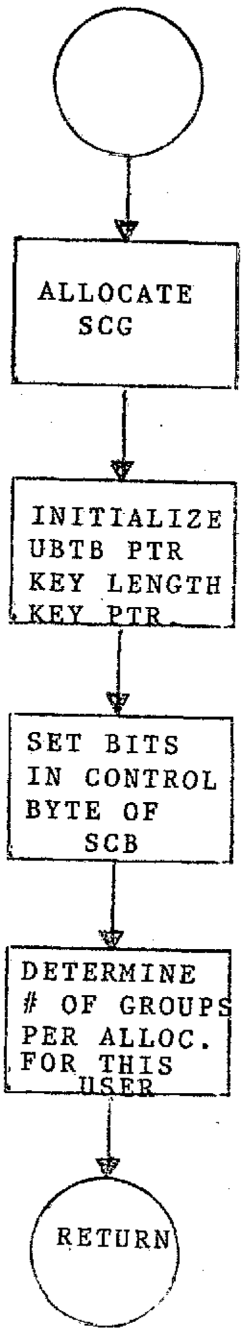

ALLOCATION ENTRY
GET ADDR OF BGN. OF MBT
GET ADDR. OF NEXT SEG TO ALLOC.
INDEX MBT BY SEC. NO.
GET ADDR. OF TRT TBL. FOR GRID1, 2, 4
WAIT
No Sec. Avail for Alloc.
TRT
End of Function
RESET ADDR TO BGN OF MAT
HIT
SET SEC.NO LOCKED
ALLOC.BITS IN MBT RSET SEC.NO.LOCK BYTE.
SET UBT. SEC.NO. ALLOC.BYTE
SET NEW SEG ADDR IN MHT
RETURN

Figure 2.2


ALLOCATE SCG
INITIALIZE UBTB PTR KEY LENGTH KEY PTR.
SET BITS IN CONTROL BYTE OF SCB
DETERMINE # OF GROUPS PER ALLOC. FOR THIS USER
RETURN

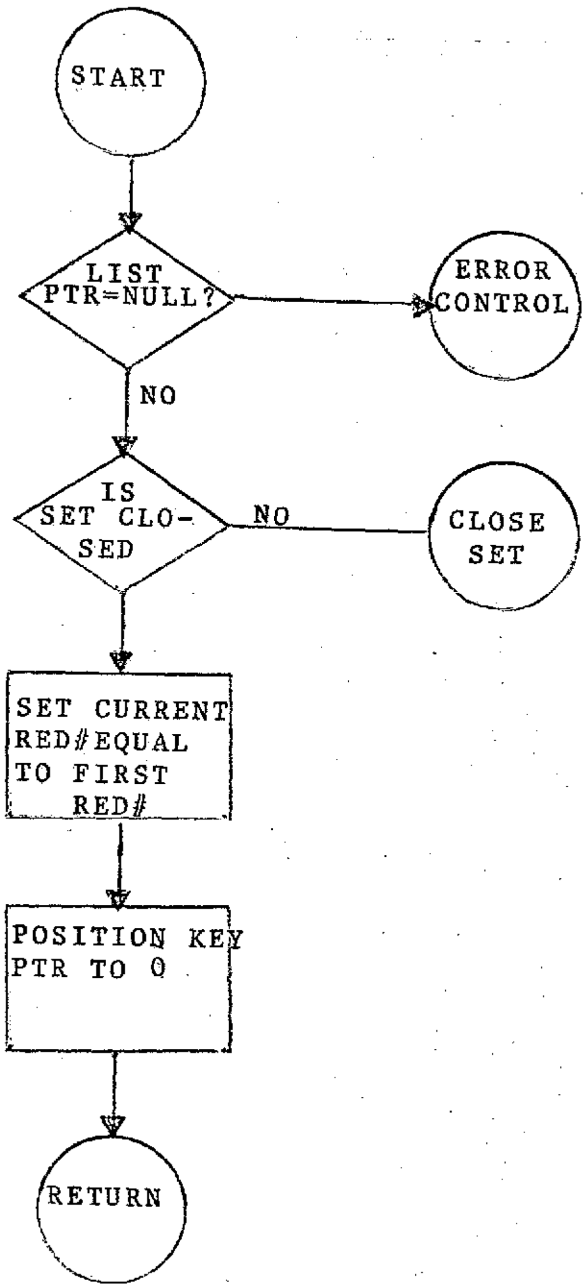


Figure 2.4 OPEN READ (Internal)

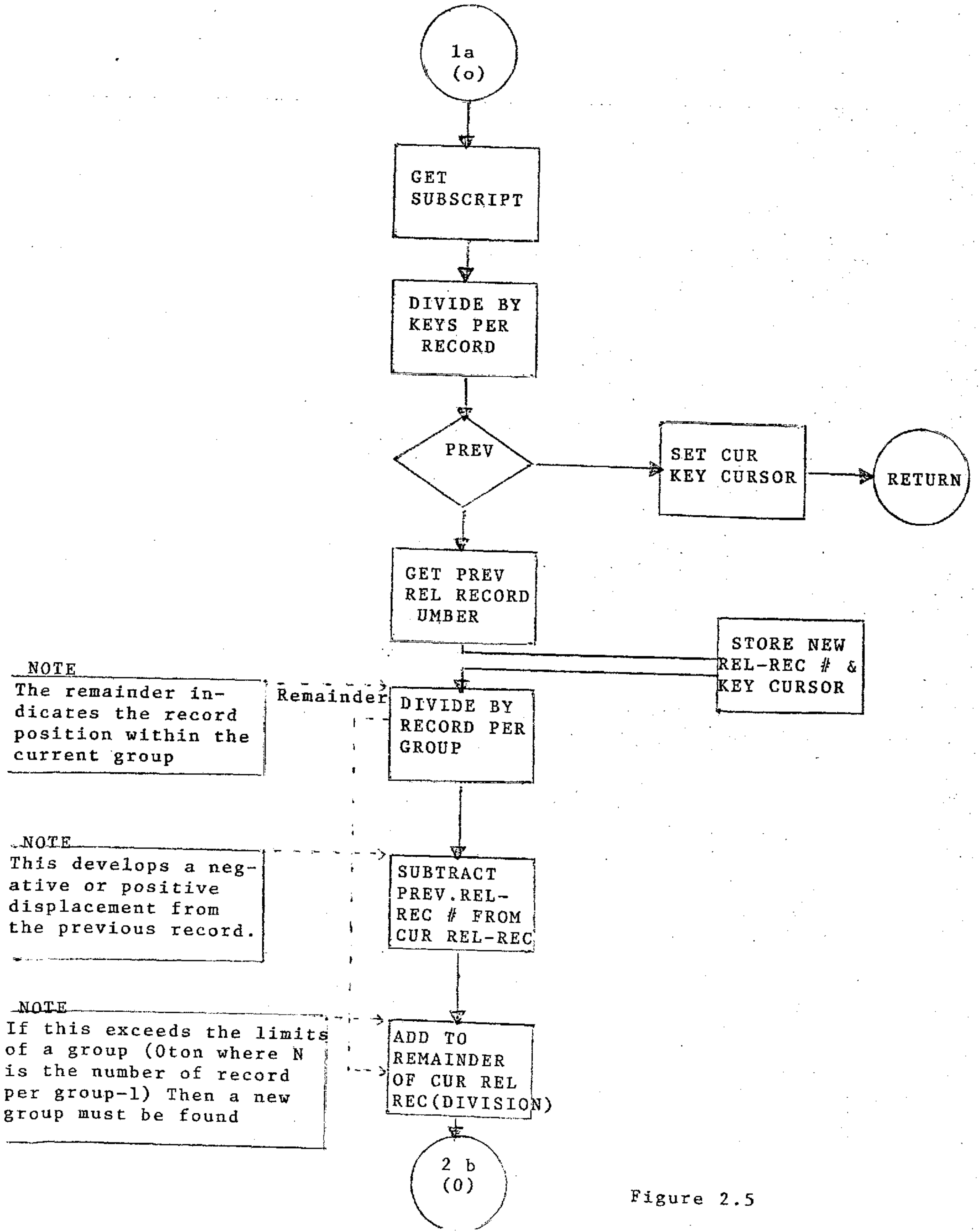

1a
(o)
GET SUBSCRIPT
DIVIDE BY KEYS PER RECORD
PREV
SET CUR KEY CURSOR
RETURN
GET PREV REL RECORD UMBER
STORE NEW REL-REC # & KEY CURSOR
NOTE
The remainder indicates the record position within the current group
Remainder
DIVIDE BY RECORD PER GROUP
NOTE
This develops a negative or positive displacement from the previous record.
SUBTRACT PREV.REL-REC # FROM CUR REL-REC
NOTE
If this exceeds the limits of a group (Oton where N is the number of record per group-1) Then a new group must be found
ADD TO REMAINDER OF CUR REL REC(DIVISION)
2 b
(0)

Figure 2.5

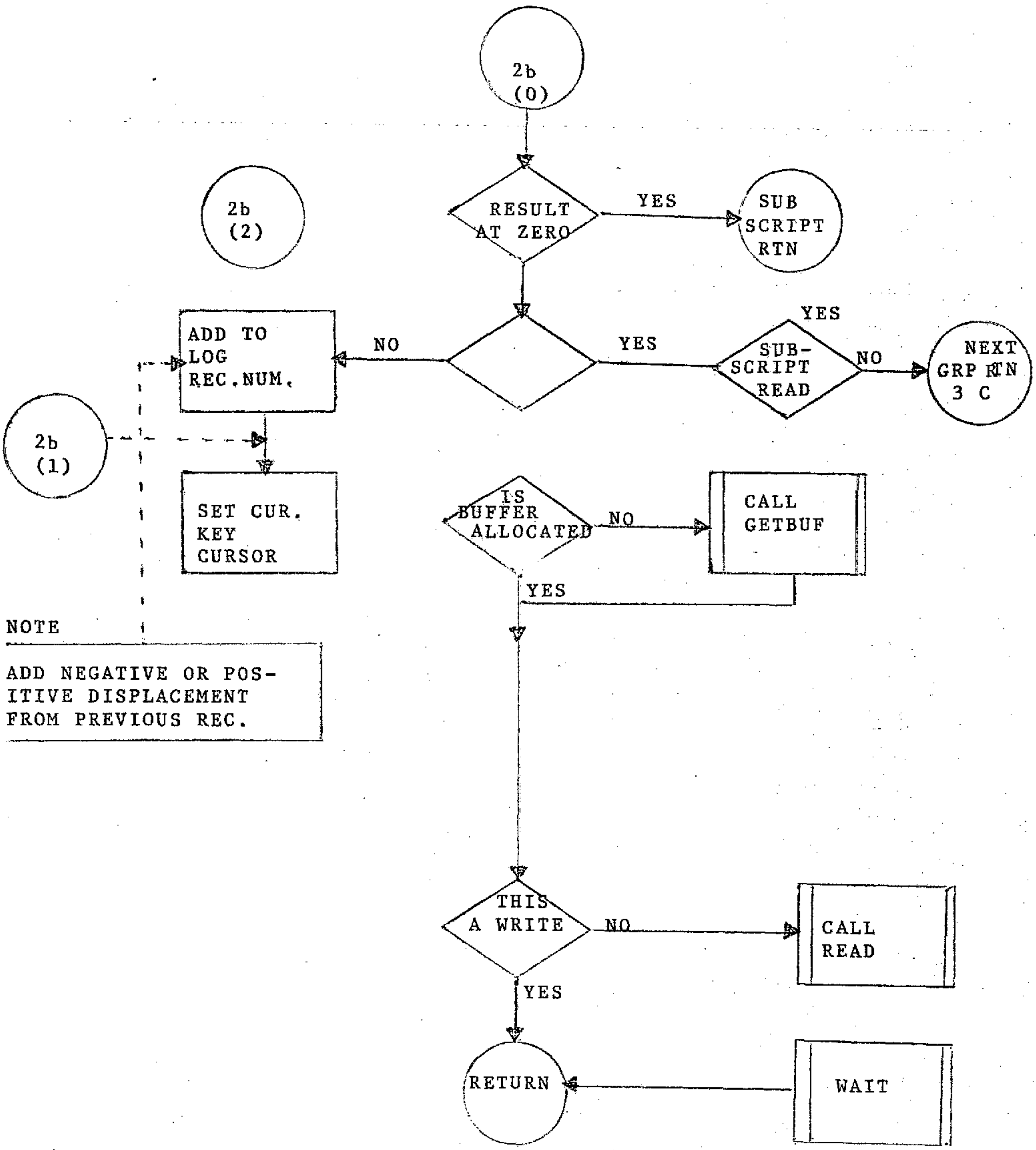

2b (0)
RESULT AT ZERO
YES
SUB SCRIPT RTN
2b (2)
ADD TO LOG REC.NUM.
NO
YES
SUB-SCRIPT READ
YES
NO
NEXT GRP RTN 3 C
2b (1)
SET CUR. KEY CURSOR
IS BUFFER ALLOCATED
NO
CALL GETBUF
YES
NOTE
ADD NEGATIVE OR POSITIVE DISPLACEMENT FROM PREVIOUS REC.
THIS A WRITE
NO
CALL READ
YES
RETURN
WAIT

Figure 2.6

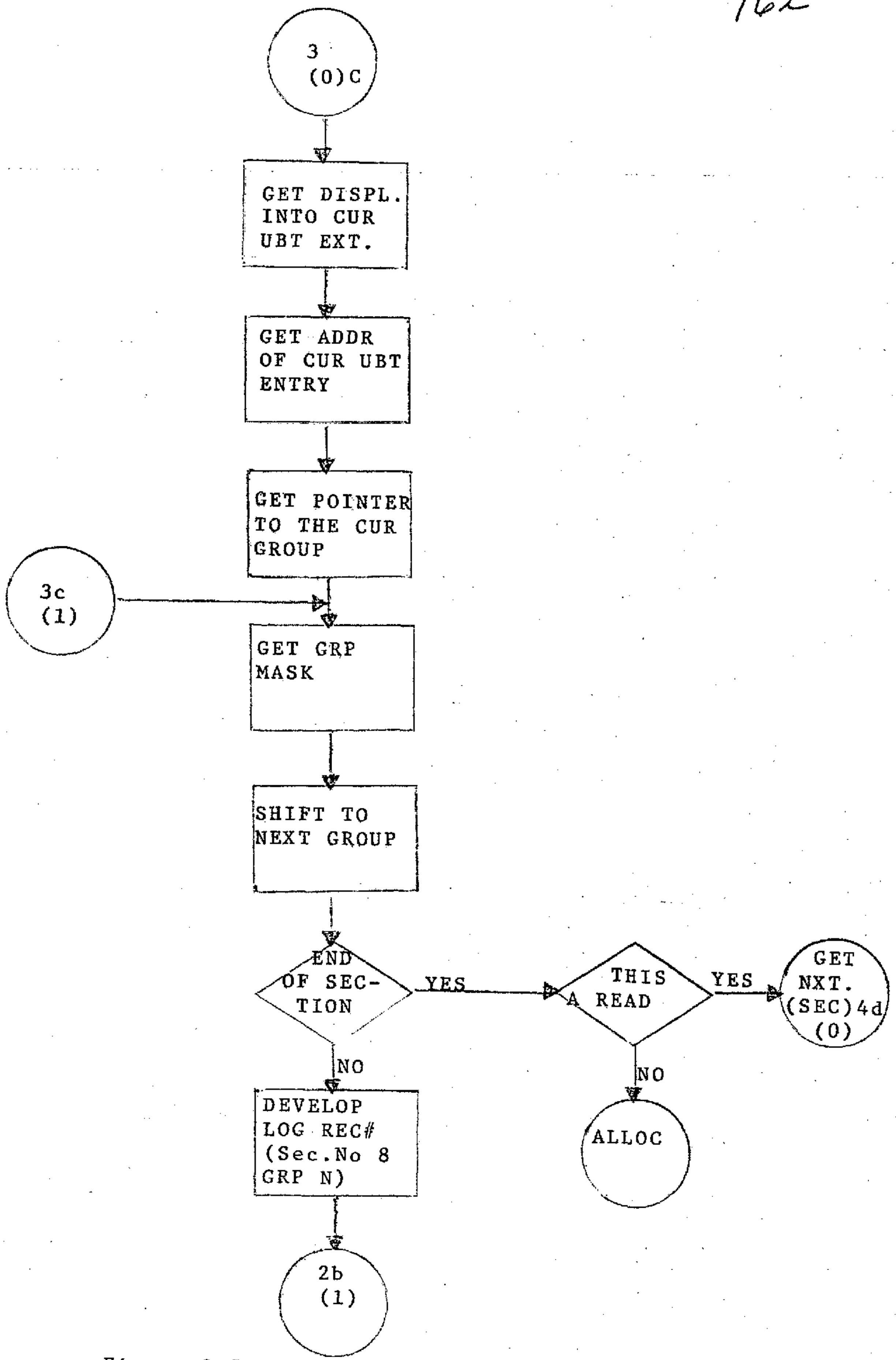


Figure 2.7 NEXT GROUP ROUTINE

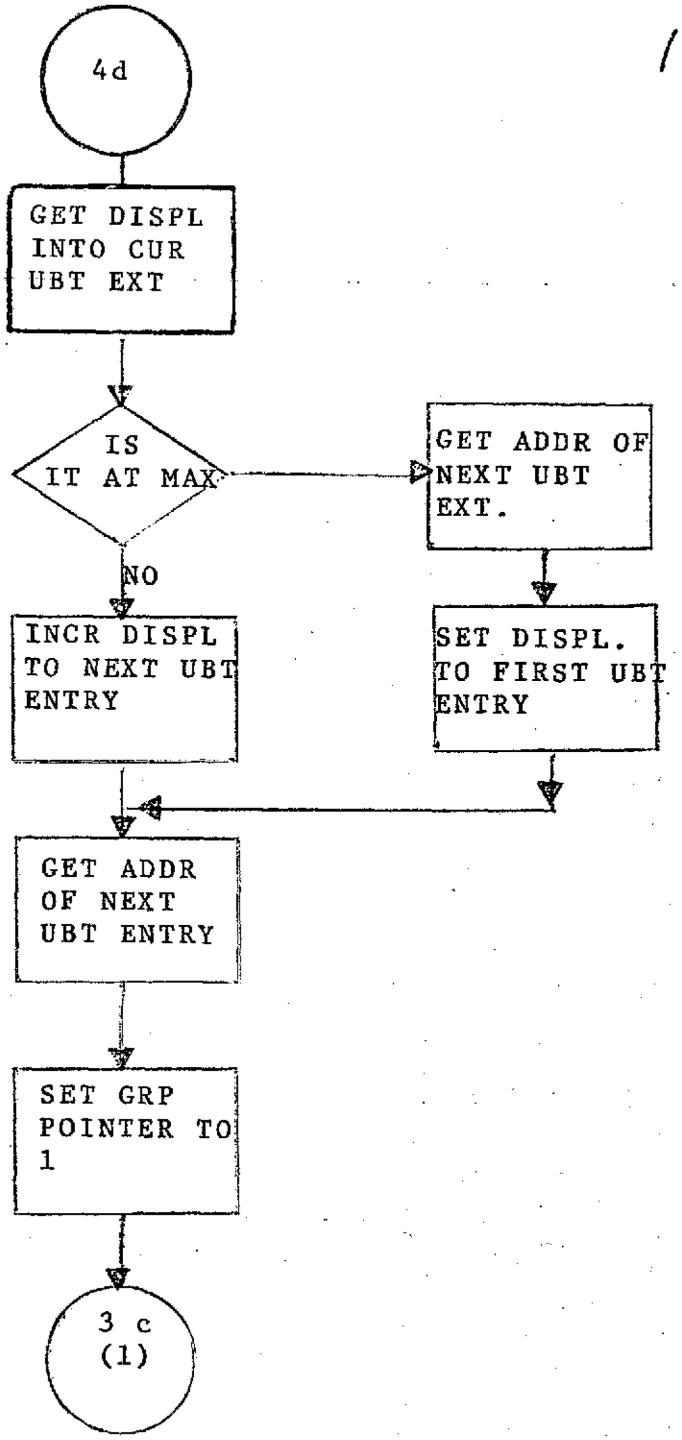


Figure 2.8 GIFT NEXT SECTION

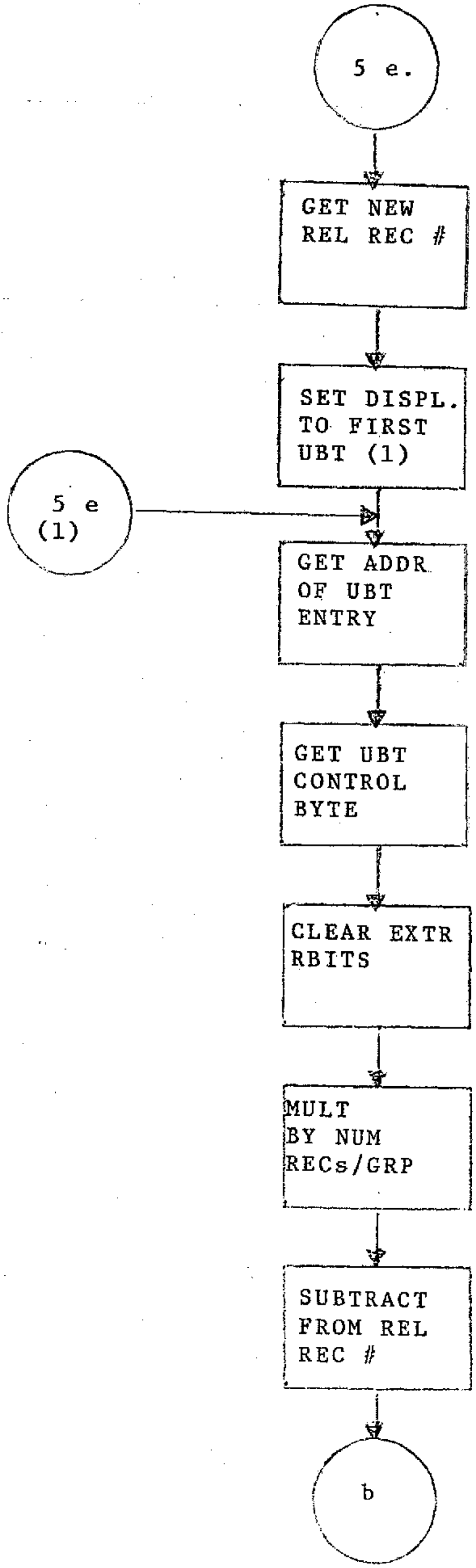

5 e.
GET NEW REL REC #
SET DISPL. TO FIRST UBT (1)
5 e (1)
GET ADDR OF UBT ENTRY
GET UBT CONTROL BYTE
CLEAR EXTR RBITS
MULT BY NUM RECs/GRP
SUBTRACT FROM REL REC #
b

Figure 2.9

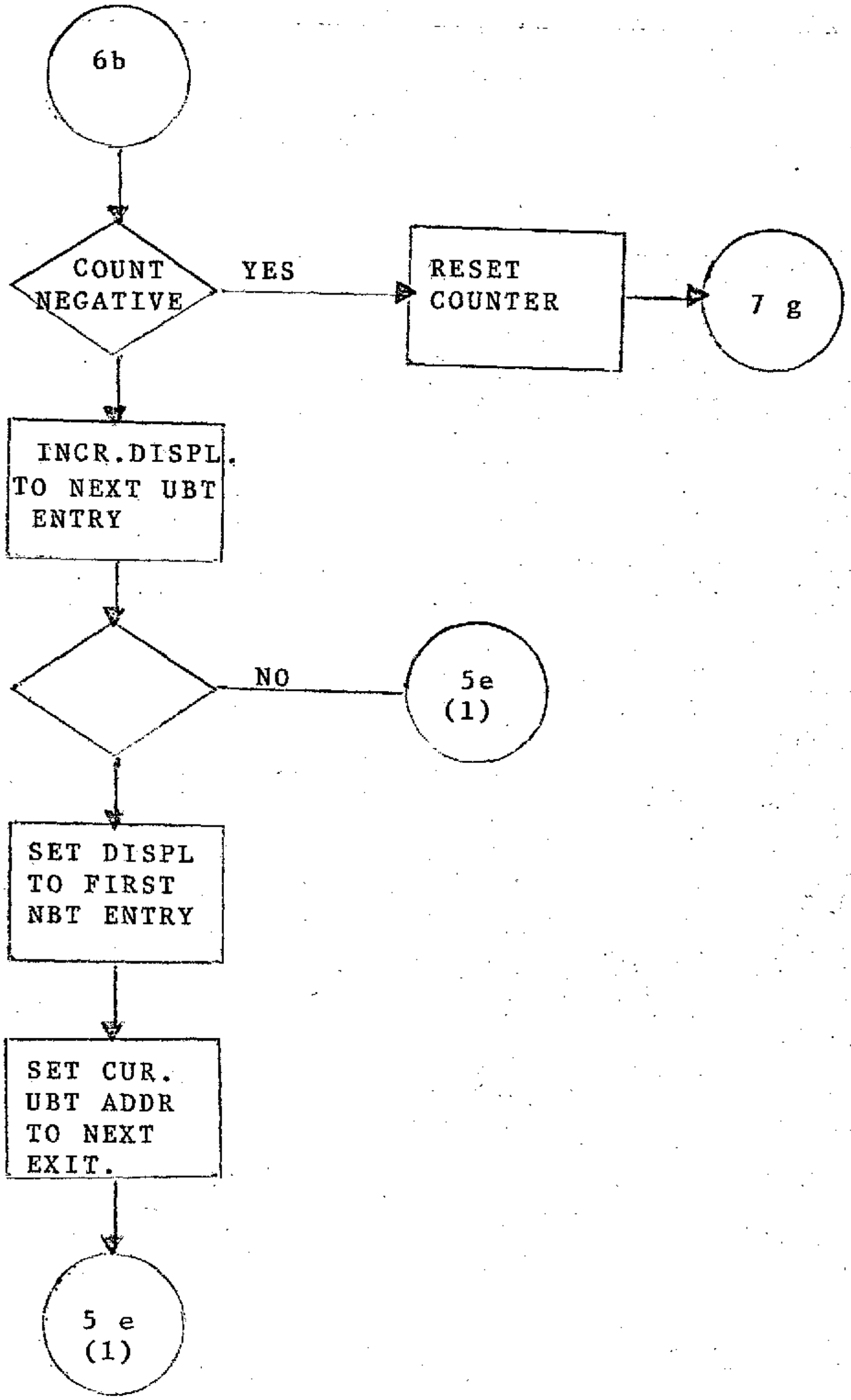
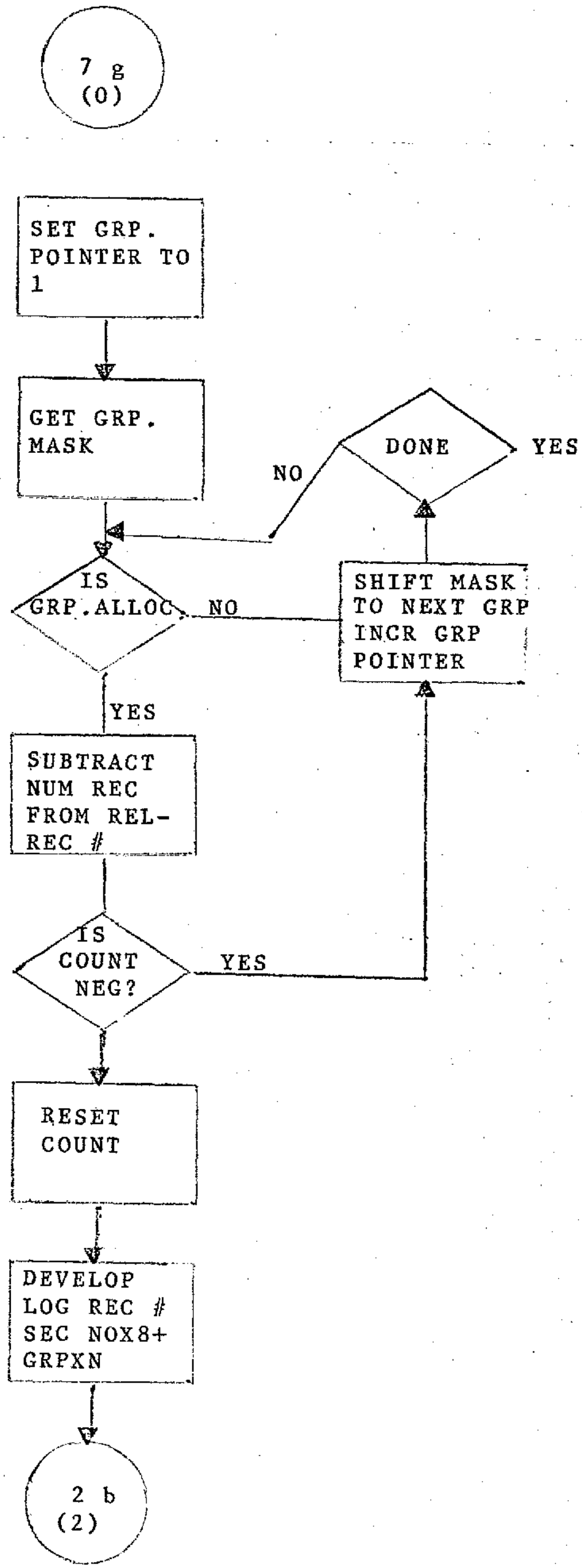

6b
COUNT NEGATIVE
YES
RESET COUNTER
7 g
INCR.DISPL. TO NEXT UBT ENTRY
NO
5e (1)
SET DISPL TO FIRST NBT ENTRY
SET CUR. UBT ADDR TO NEXT EXIT.
5 e (1)

Figure 2.10



7 g
(0)
SET GRP.
POINTER TO
1
GET GRP.
MASK
DONE
YES
NO
IS
GRP.ALLOC
NO
SHIFT MASK
TO NEXT GRP
INCR GRP
POINTER
YES
SUBTRACT
NUM REC
FROM REL-
REC #
IS
COUNT
NEG?
YES
RESET
COUNT
DEVELOP
LOG REC #
SEC NOX8+
GRPXN
2 b
(2)

Figure 2.11

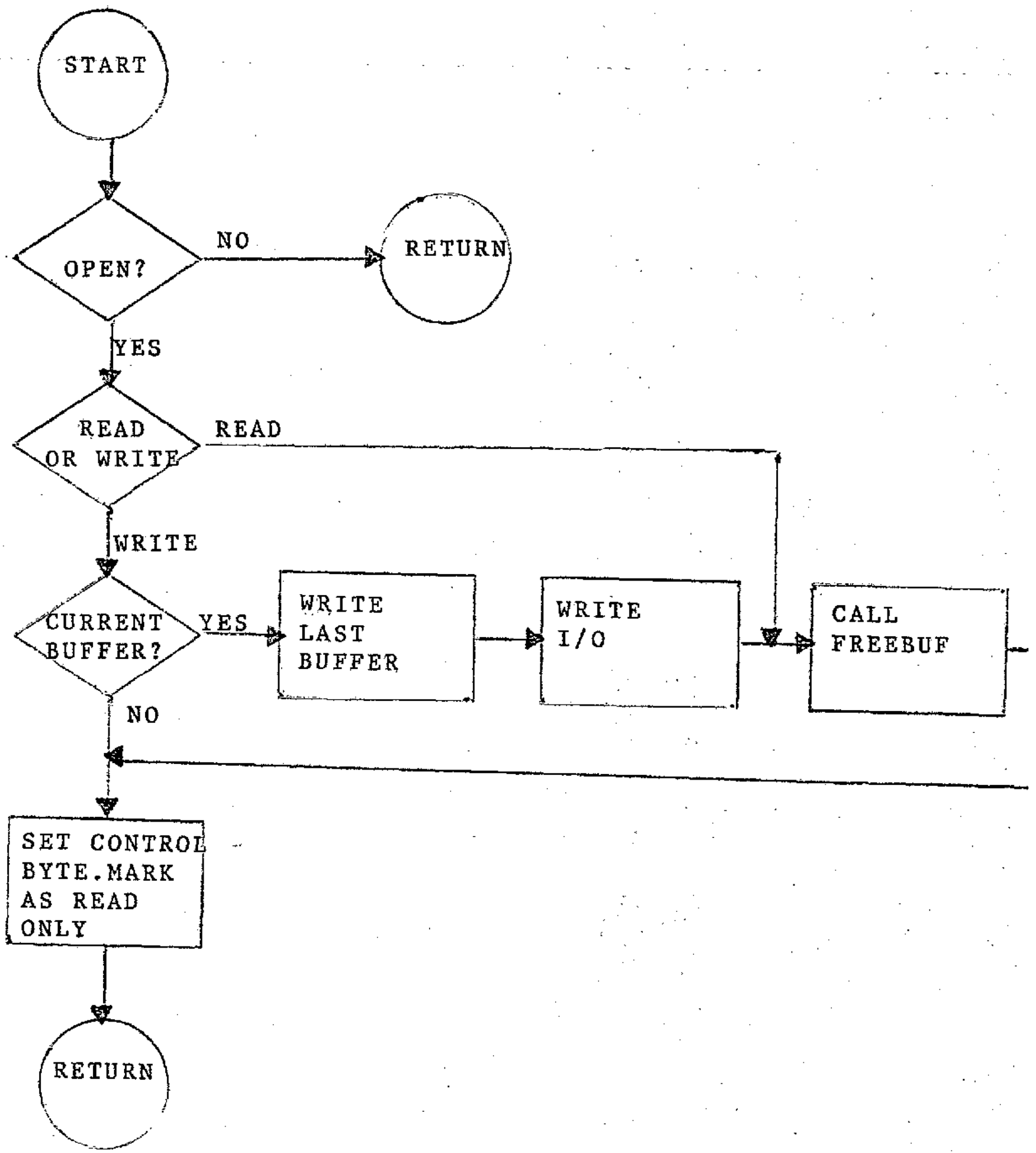


Figure 2.12 CLOSE (Internal)

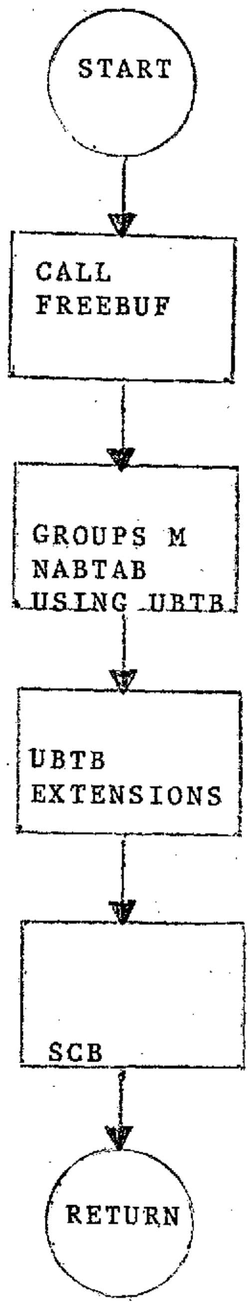


Figure 2.13 ERASE (EXTERNAL)

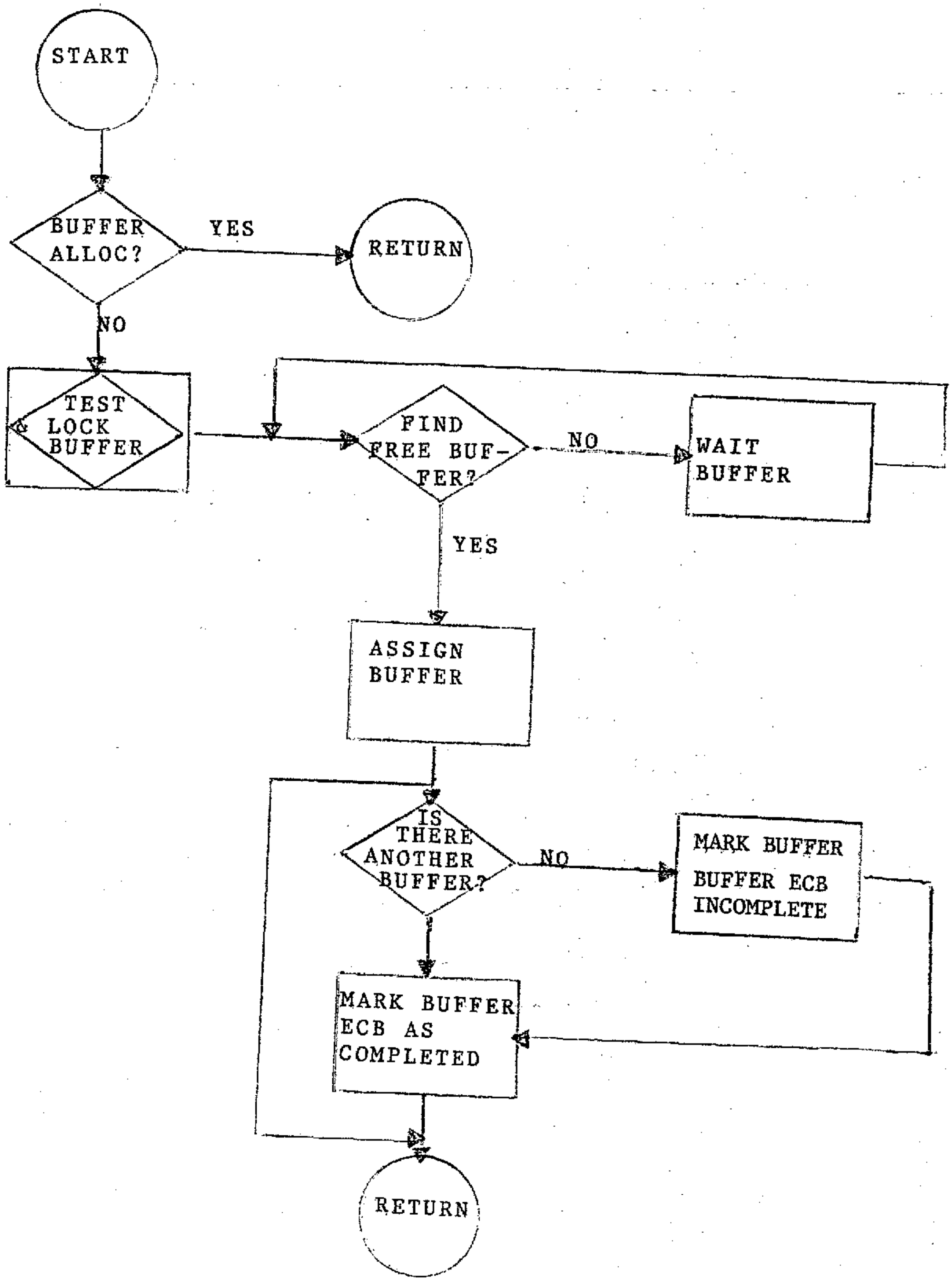


Figure 2.14 GETBUF (Internal)

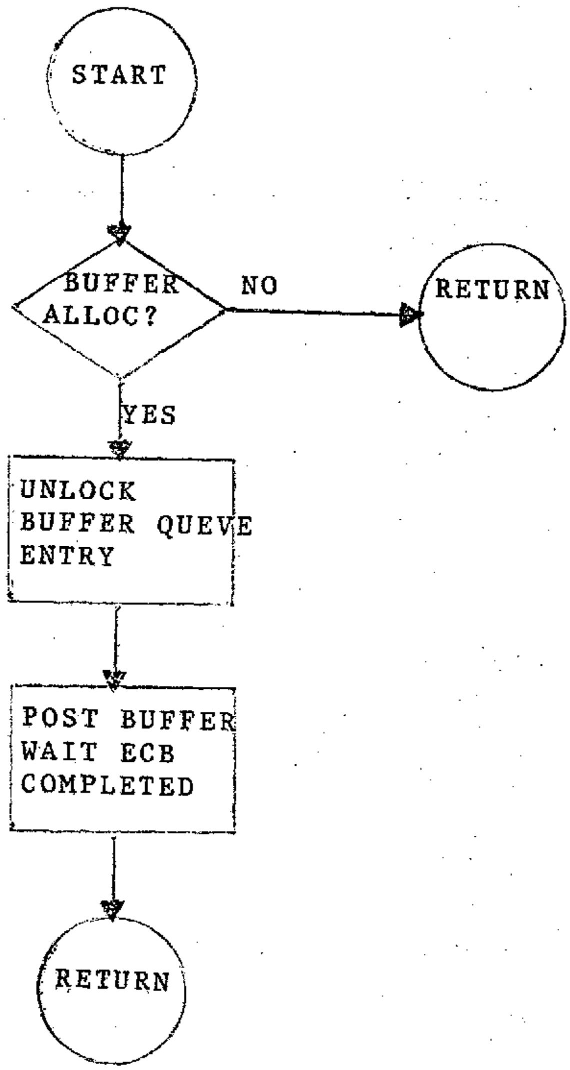


Figure 2.15 FRFTPUT (Internal)

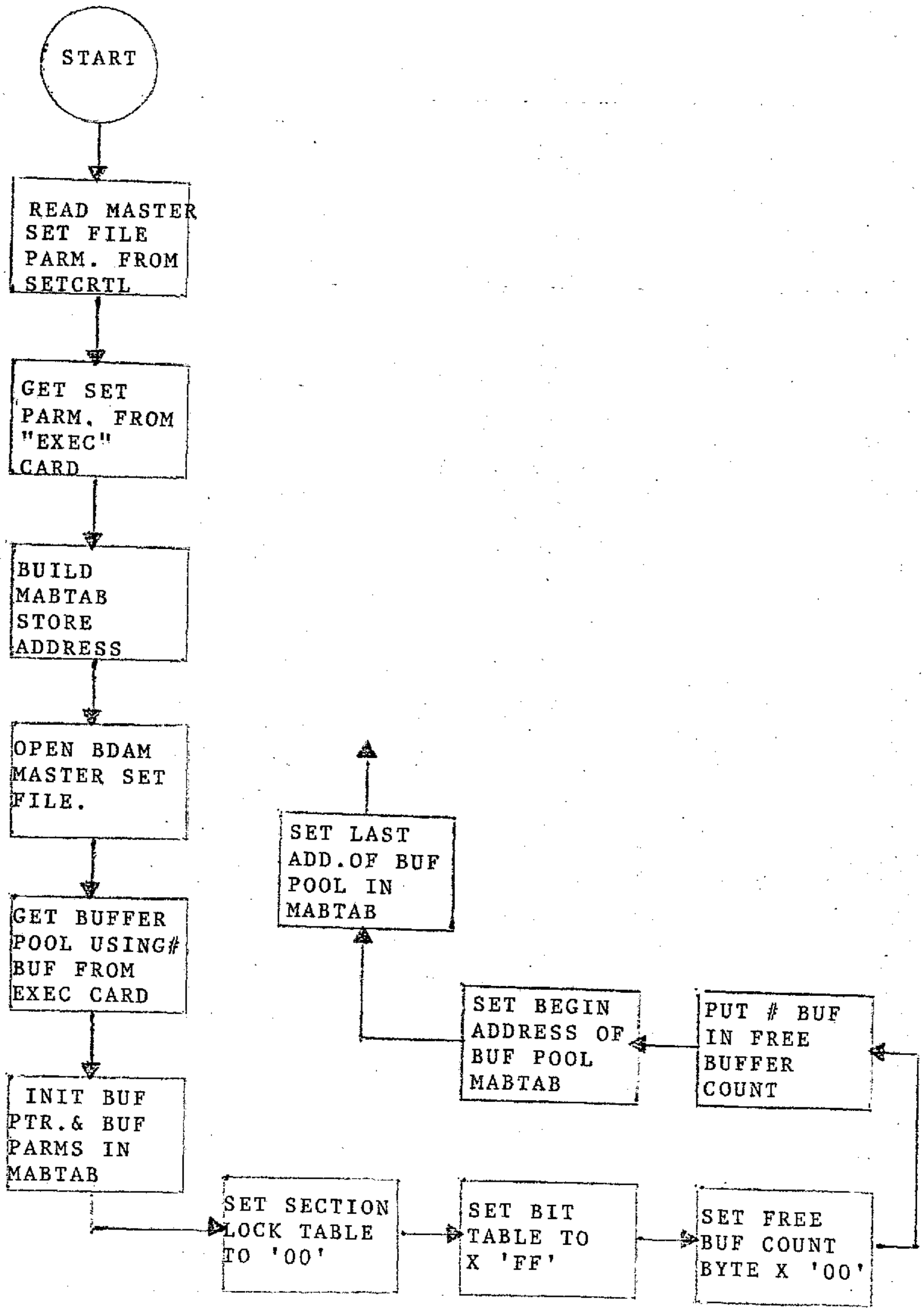


Figure 2.16 SYSTEM LOAD

TOPIC B.9 - SET FILE I/O

A. MODULE NAME:

Program-ID - NDBSETIO
Module-ID - DBSETIO

B. ANALYST

Tom C. Moser
Neoterics, Inc.

C. MODULE FUNCTION

This program is called by the NASIS Set Manager DBOSET to access the set file. Passed in register one will be the address of the parameter list containing a word for the address of the DECB, a word for the address of a buffer, and a word containing the actual relative record number to read/write.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1.

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

   c. Input Files

      The Set File is accessed by the program to write or read a record of set items.

   d. On-Line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      The Set File is accessed by the program.

   b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

After initial housekeeping, this module checks to see if the setfile is open. If not, it is opened. After the open of the setfile, the DCB is checked to see if the open completed successfully. If not, an error code is set into R15 and the module returns.

If upon entry to this module the DCB is already open or after a successful open to the DCB, the parm list passed is validated for proper data. Depending on the bit settings in the DECB, a read or write is performed to the setfile using the parameters passed.

F. CODING SPECIFICATIONS

1. Source Language

The module is written in IBM Assembler language.

2. Suggestions and Techniques

Not Applicable

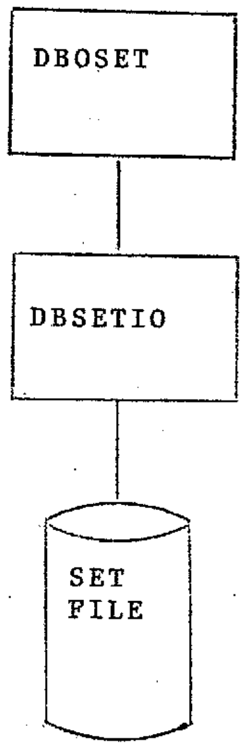


Figure 2. I/O Block Diagram

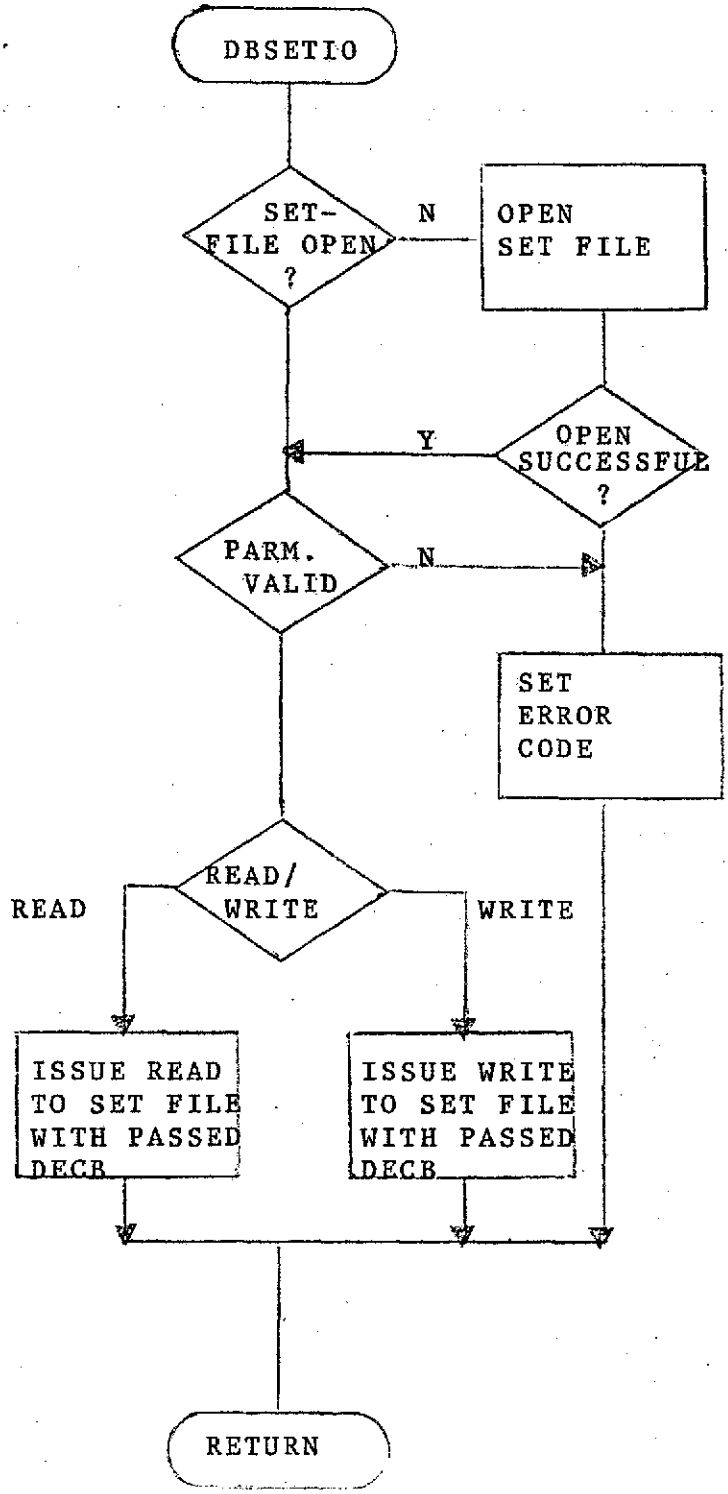


Figure 2. TOP LEVEL FLOWCHART

TOPIC B.10 - FIELD UTILITIES

A. MODULE NAME:

Program-ID: NDBFLDU
Module-ID: DBFLDU

B. ANALYST

William Petrarca

C. MODULE FUNCTION

NDBFLDU supports retrieval modules requiring various field-related functions previously provided with the FLDTAB table of NASIS/TSS Release 2. The four basic functions are described in the following paragraphs, each representing an entry point.

1. Field Classification - FLDCLAS

   This function returns pertinent information about a particular field. The field is classed as to (1) invalid, (2) anchor-resident, (3) associate-resident, (4) subfile-resident. Furthermore, the subfile character of a subfile or control field is provided, along with a bit switch indicating an indexed field.

2. Find control field name-FLDCTRL

   This function returns the name of the control field for a provided subfile character or subfile field.

3. Find key field name-FLDSKEY

   This function provides the name of a key field for either the anchor file or a subfile.

4. Get sequential-format field name-FLDGET

   This function returns the next field name in the predefined sequential formats 2 thru 5. The calling routine provides the subscript field number.

   This function has an inverse capability of providing a subscript field number for a given field name.

   All entry points will be called via PL/I conventions.

D. DATA REQUIREMENTS:

1. I/O Block Diagram

   Not Applicable.

2. Input Data Sets

   Not Applicable

3. Output Data Sets

   Not Applicable

4. Reference Tables

   a. Mainline File Control Block

      (MFCB) is setup by DBPAC and contains database file status switches and pointers to the descriptor table and the file control block (FCB) for each file. NDBFLDU merely references this block to return the proper field information.

   b. Descriptor Tables (DESC) are the internal file descriptors; these tables contain information concerning each field on each data base file. DBPAC links descriptor tables' field elements consecutively, similar to the fields sequential order in the original FLDTAB with a pointer named DESC.FLD.FCP which points to the next DESC.FLD in the chain. NDBFLDU utilizes this linkage for the FLDGET function.

   c. Field table (OS version)-FLDTAB contains basic information concerning the fields of a data base. NDBFLDU references FLDTAB.FIELD.# to verify the total number of fields on the data base.

      Please refer to the Data Set Specifications for each of the individual tables for details.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 1

2. Narrative

a. FLDCLAS

This routine looks for a DESC.FLD.FLDNAME in the file descriptors identical to the parameter field name. When it is found, DESC.FLD. GROUP reveals the file of residence. Super-fields have DESC.FLD.NAMECNT greater than zero. The DESC.FLD.ASSOCFIL byte flags associate file fields. The DESC.FLD.SUBCNTRL byte is set on for control fields and, likewise for inverted fields with the DESC.FLD.INVFILE byte. Consequently, the parameters are set as follows:

parm 1: the MFCB
parm 2: 0 - invalid field
1 - anchor field
2 - associate field
3 - subfile field
parm 3: the field name.
parm 4: either a subfile field character for subfile fields and control fields or a blank for non-subfile fields.
parm 5: a bit on for indexed fields.

b. FLDSKEY

This routine searches for the file descriptor whose MFCB.FILE_NAME ends with the supplied subfile character. Once found the file descriptors contain the key field name in DESC.FLD(1).FLDNAME. Consequently, the parameters are set as follows:

parm 1: the MFCB
parm 2: 0 - unsuccessful call
1 - successful call
parm 3: the key field name
parm 4: the file suffix character

c. FLDCTRL

If a field name is given, this routine performs an internal field classification to get the field names subfile suffix; if a subfile suffix is provided. this routine proceeds as normal. The field descriptors for the anchor file are searched until DESC.FLD.SUBCNTRL is on and DESC.FLD.SUBFILE is the same as the subfile suffix. Consequently, the parameters are set as follows:

parm 1: the MFCB
parm 2: 0 - unsuccessful call
1 - successful call
parm 3: the control field name for the subfile.
parm 4: the suffix of a subfile.

d. FLDGET

This routine performs two functions. By providing a field subscript number to the standard format 4 list of fieldnames, the caller can obtain the particular field's name in any order. Inversely, by providing a valid field name, the caller can get the particular field's subscript number in the format 4 list.

For the former function this routine loops down the DESC.FLD.FCP pointer chain the number of fields specified in the given field number; the final DESC.FLD.FLDNAME is returned. For the latter function the same chain is followed with a comparative check for the given field name; having been found, the subscript number is returned.

parm 1: the MFCB
parm 2: the field number
parm 3: the field name

F. CODING SPECIFICATIONS

1. Source Language

The NDBFLDU coding is to employ the IBM PL/I programming language.

2. Suggestions and Techniques

NDBFLDU is not to alter any MFCB or related tables to perform its functions so as to guard the integrity of DBPAC.

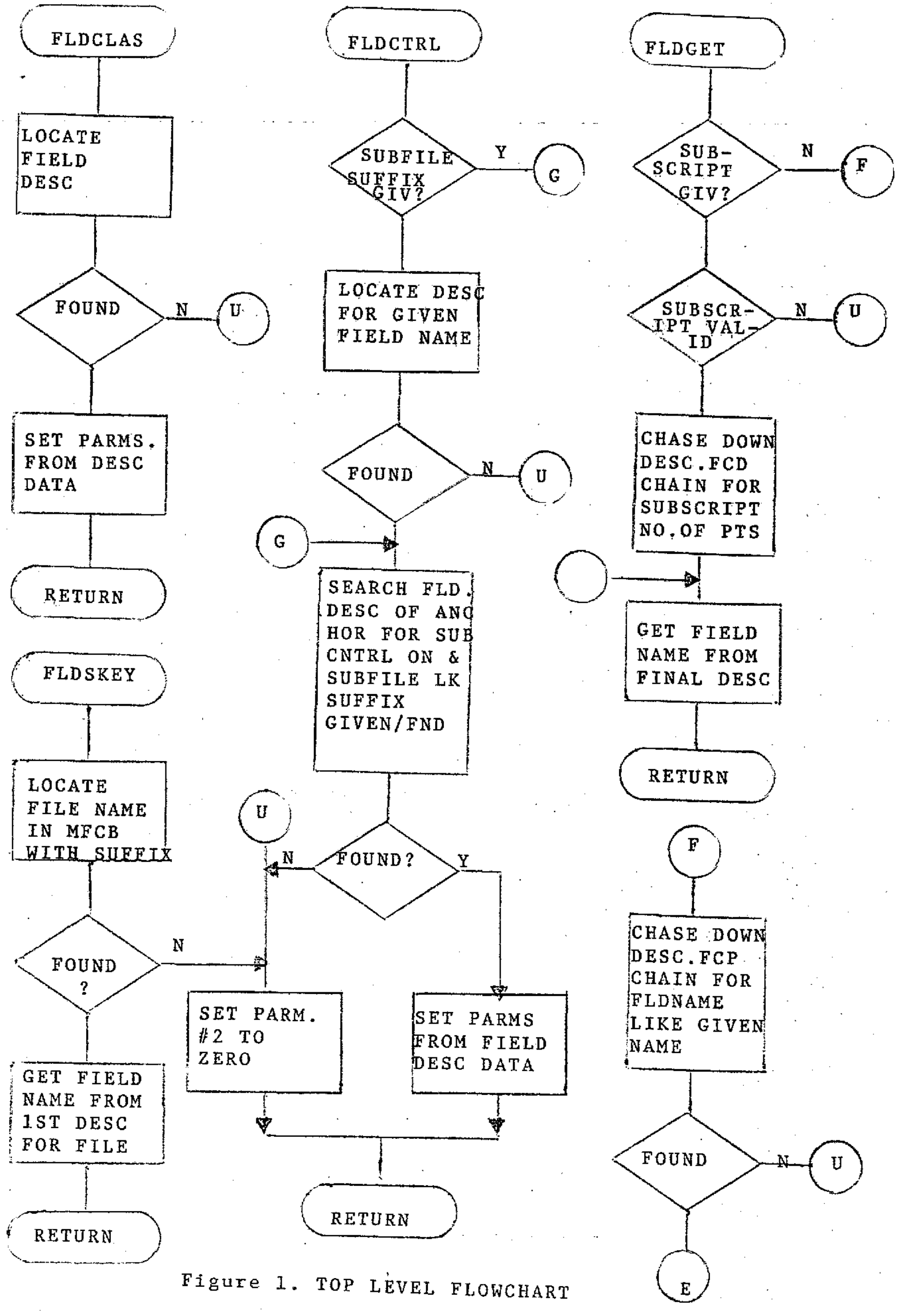


Figure 1. TOP LEVEL FLOWCHART

TOPIC B.11 - GENERAL CALL BY NAME

A. MODULE NAME:

Program-ID - NDBCALL
Module-ID - DBCALL

B. ANALYST

Tom Moser, William Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

This module allows a PL/1 module to call an external entry point by specifying its name at execution time. Any parameters other than the entry point name will be passed on to the called entry point. The name specified must conform to the construction standards of OS/360.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The ESDTAB file is read in to build an in-core table of entry-points and their addresses; ESDTAB is created by the stand-alone DBTABLE module.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

Not Applicable

4. Reference Tables

TRQDSECT - Terminal Request Table
METDSECT - Module Entry Table

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry the program checks for first time entered. If this is the first time the program is entered, an initialization of tables is begun. Otherwise, the table initialization is bypassed.

The first table built is an in-core duplicate of the ESDTAB data set which has been sorted by DBTABLE, a stand-alone program. The table contains all the external symbols of the NASIS load module in alphabetic order along with the relative offset in the module and the segment number the external symbol resides within. A segment greater than 1 implies a transient entry point.

The second table built is a quick index of first letters into the first table built. When this table is built, initialization is complete.

The requested entry point is now looked for in the in-core ESDTAB. The index table is used to position the first ESDTAB entry to begin searching.

If none is found, a diagnostic is sent to the OS operator and a return is made to the calling program.

Upon finding the requested entry point, its segment number is checked to determine transiency.

A non-transient module can be called (branched to) immediately; therefore, the registers are set up for any parameters and the call is made. Afterwards, the caller is returned to.

A transient module must be made resident before being called. This is accomplished by putting

the entry point to be loaded into TRQMODRQ of the TRQDSECT and calling the MTTCALL entry in the monitor (MTTSUP). Upon return the requested module will have been loaded and its address will be in METENTRY of the METDSECT pointed to by register 9.

This program also maintains the number of users for any loaded transient module. To do this the METUSERS field in METDSECT is incremented before the control is passed to it.

The registers are set up for any parameters and the call is made.

Upon return from the transient module the METUSERS field is decremented. Furthermore, the TRQMODRQ field is blanked out to show no transient activity for this terminal.

Finally the caller is returned too.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in IBM Assembler language for speed and efficiency.

2. Suggestions and Techniques

The initialization process must be uninterruptable to prevent reentrance during its occurrence; the MTTMUST function should be employed to facilitate this.

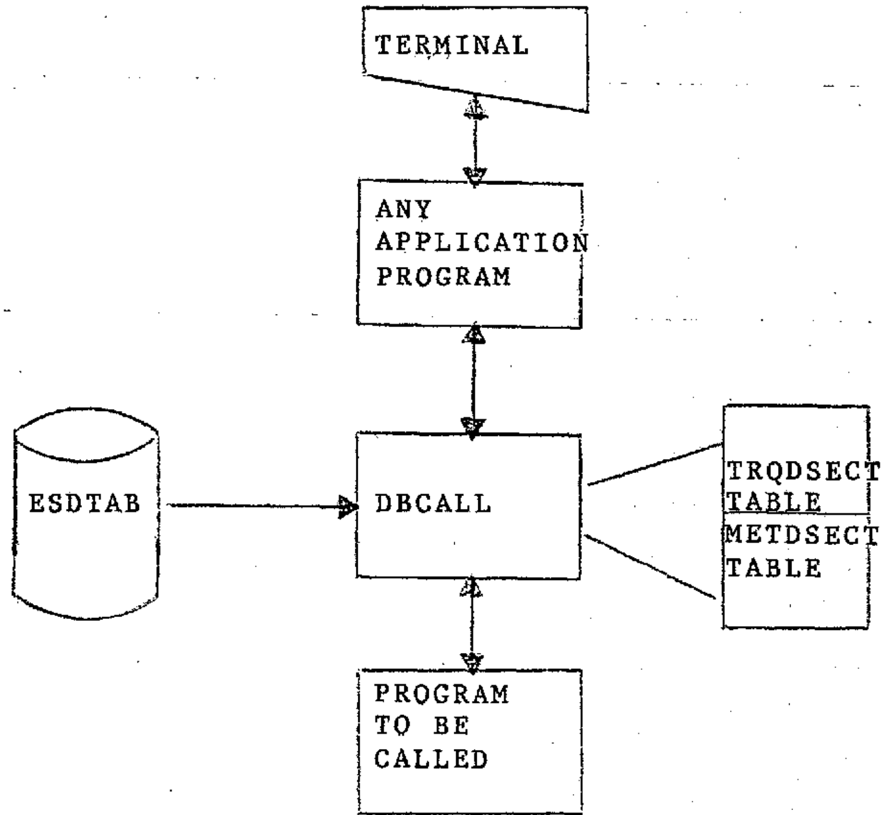


Figure 1, I/O BLOCK DIAGRAM

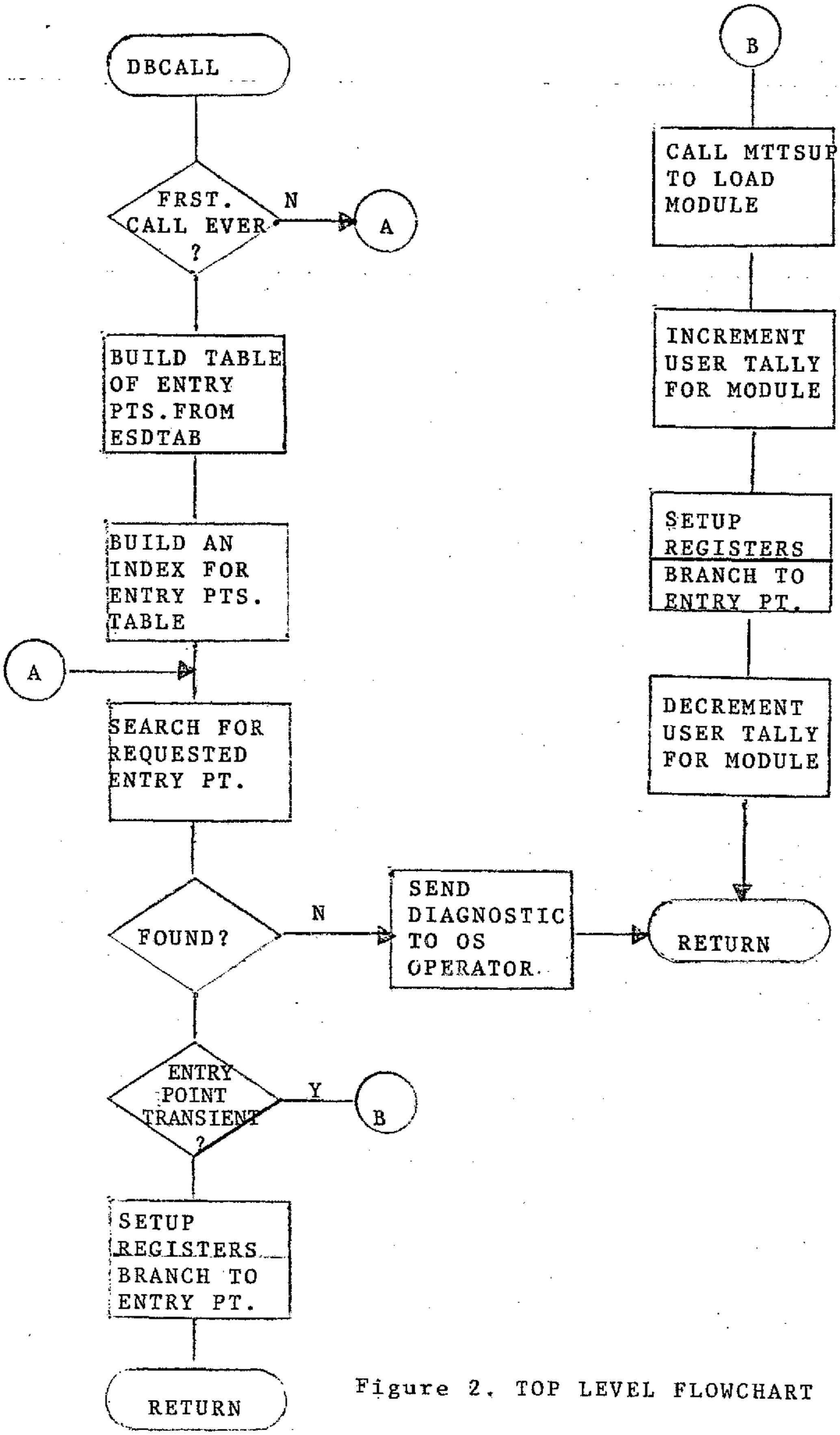


Figure 2. TOP LEVEL FLOWCHART

TOPIC C.1 - UTILITIES JOIN (RDBJOIN)

A. MODULE NAME

Joining new NASIS users
Program-ID - NDBJOIN
Module-ID : DBJOIN

B. ANALYST

Edward J. Scheboth, Jr.
Neoterics, Inc.

C. MODULE FUNCTION

This program gives the NASIS DBA the ability to create and maintain the data set NASIS.USERIDS which contains the NASISIDS under which users of the NASIS system are given access to MT/T, the Retrieval system and the various dataplices. The data set NASIS.USERIDS is organized under ISAM, and has as a key the eight byte NASISID of each joined user, with a variable record format containing his password, timeslice, user authority, and list of permitted files.

This program has as a secondary function the task of displaying for DBINIT the files available for retrieval to a specific user.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data sets

a. Parameter cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The NASISIDS data set. (For complete detailed specifications of this file see Section III of the Development Workbook).

d. On-line Terminal Entries

Valid JOIN commands.

3. Output Data Sets

   a. Output Files

      See 2.c

   b. On-line Terminal Displays

      See 2.d

   c. Formatted Print Outs

      Not Applicable

   d. Punched Card Output Files

      Not Applicable

4. Reference Tables

   Not applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   The primary entry point of this program (DBJOIN) is responsible for maintenance and display of the NASIS.USERIDS file. It is necessary that this file be safeguarded from tampering and it therefore should be password protected.

   The main routine should have a prompt validation loop which calls the subordinate functions such as Join, Quit... etc. making the program more modular and much easier to modify. To enhance speed of the system these calls should not be to internal procedures but should be pseudo calls set up using the extensive facilities of the PL/1 preprocessor to simulate these calls.

   Program termination should be thru the common END convention set up in TS/2. All parameters to the commands shall be obtained using the new TS/2 facilities.

   The proper value of the first parameter signifies

this module is to display the available files for DBINIT. This is really a sub function of the main routine's Display function and paging entry and should be coded as such to facilitate coding.

F. CODING SPECIFICATIONS

1. Source Language

As much as possible of the RDBJOIN module is coded in the IBM PL/1 programming language. The input and output coding for accessing the file NASIS.USERIDS is handled by a direct call to the DBPAC assembler routines. All terminal access is handled by TS/2.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications.

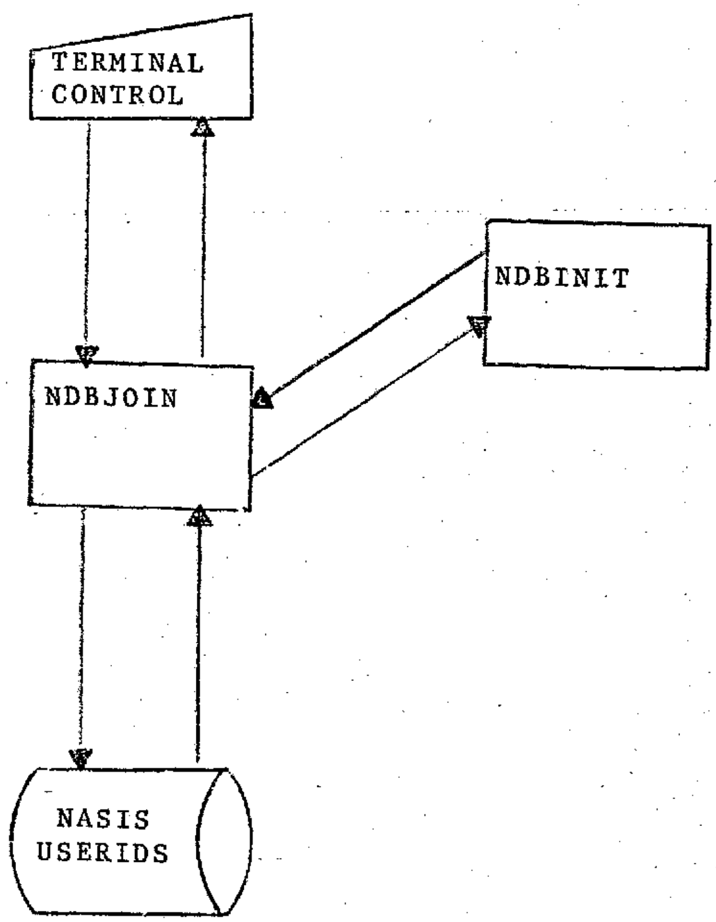


Figure 1. I/O BLOCK DIAGRAM

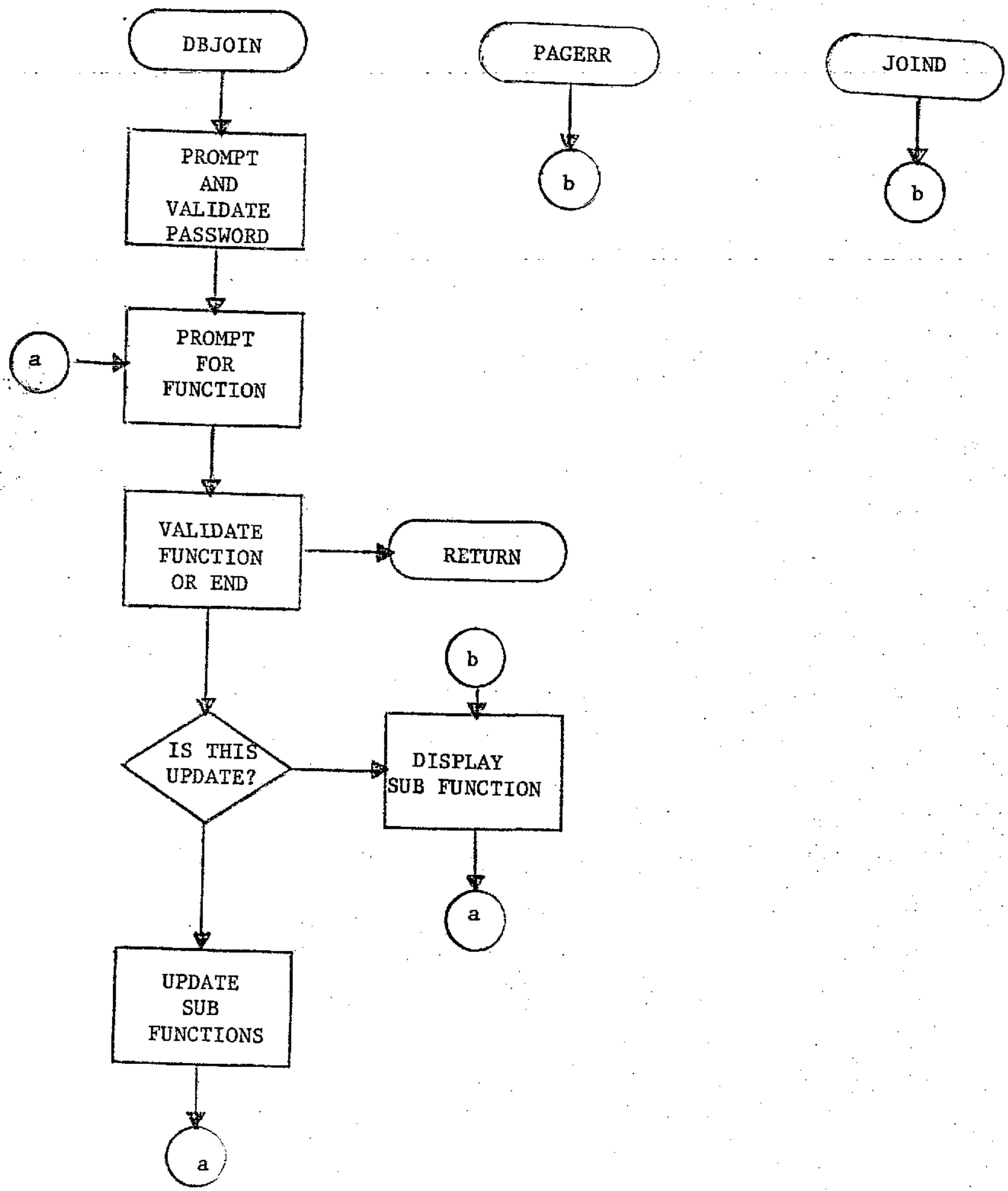


Figure 2. Top Level Flowchart

TOPIC C.2 - ESD TABLE GENERATOR

A. MODULE NAME

Program-ID: NDBTABLE
Module-ID: DBTABLE

B. ANALYST

Tom C. Moser, William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

This module reads the composite external symbol dictionary (CESD) of the NASIS load module to generate an ordered ESDTAB file to be used by DBMTAB (to generate the MODTAB file) stand-alone and by DBCALL (for general call by name) of execution time.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1.

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      1. DDNAME MODULE: This file is the load module from which the CESD is to be processed.

      2. DDNAME SORTIN: This file is built by this module from the CESD; it contains the external symbol name, its segment number, and its relative offset. DBTABLE then calls the OS Sort facility to sort this file. After the sort this file is rewritten to form the ESDTAB file.

      3. DDNAME SORTOUT: This file is the output file from the OS Sort facility. It is

read by DBTABLE following the Sort.

4. Sort work files: These files are used by the IBM OS Sort utility. Refer to the IBM Utilities Manual for details.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

1. DDNAME SORTIN: This file is initially an output file. After being processed by the Sort facility, this file is rewritten to become the final ESDTAB file.

2. Sort work files used by utility.

3. SYSPRINT: Sort Diagnostics.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top level flowchart

See Figure 2.

2. Narrative

On entry DBTABLE opens the MODULE and SORTIN files and writes a dummy (zeroes) record to the SORTIN file; this record will be used later for control information in ESDTAB. Then the load module is read to get all ESD-type records. Each ESD-type record is written to SORTIN. Having read all

needed records, DBTABLE attaches the Sort utility to sort the external symbols in the ESD records. When the sort is finished, SORTOUT file has the sorted ESD records. This file is read into core. The segment and offset information are posted into an ESDTAB record with each external symbol name, rendering a 32-byte ESDTAB record - 8 bytes of name, 8 bytes of data, and 16 bytes of zeroes. The SORTIN file is re-opened for output; the first record written is 8 bytes of zeroes, followed by 4 bytes of total records in the ESDTAB file, followed by 4 bytes of total segment 1 csects found in the CESD, and followed by 16 bytes of zeroes. Then all the ESDTAB records are written from core.

F. CODING SPECIFICATIONS

1. Source Language

DBTABLE is written in OS/360 Assembler language.

2. Suggestions and Techniques

Not Applicable

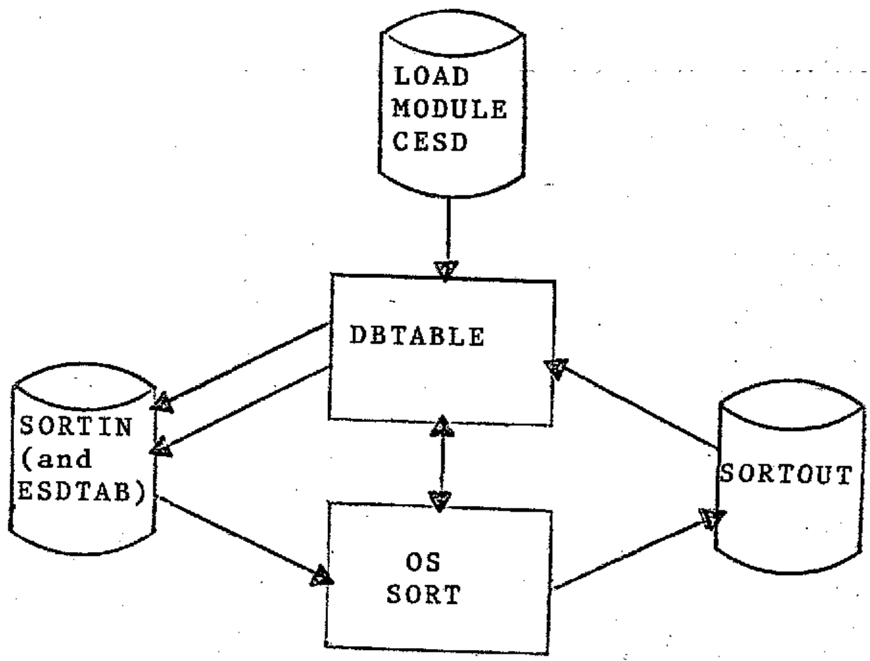


Figure 1. I/O BLOCK DIAGRAM

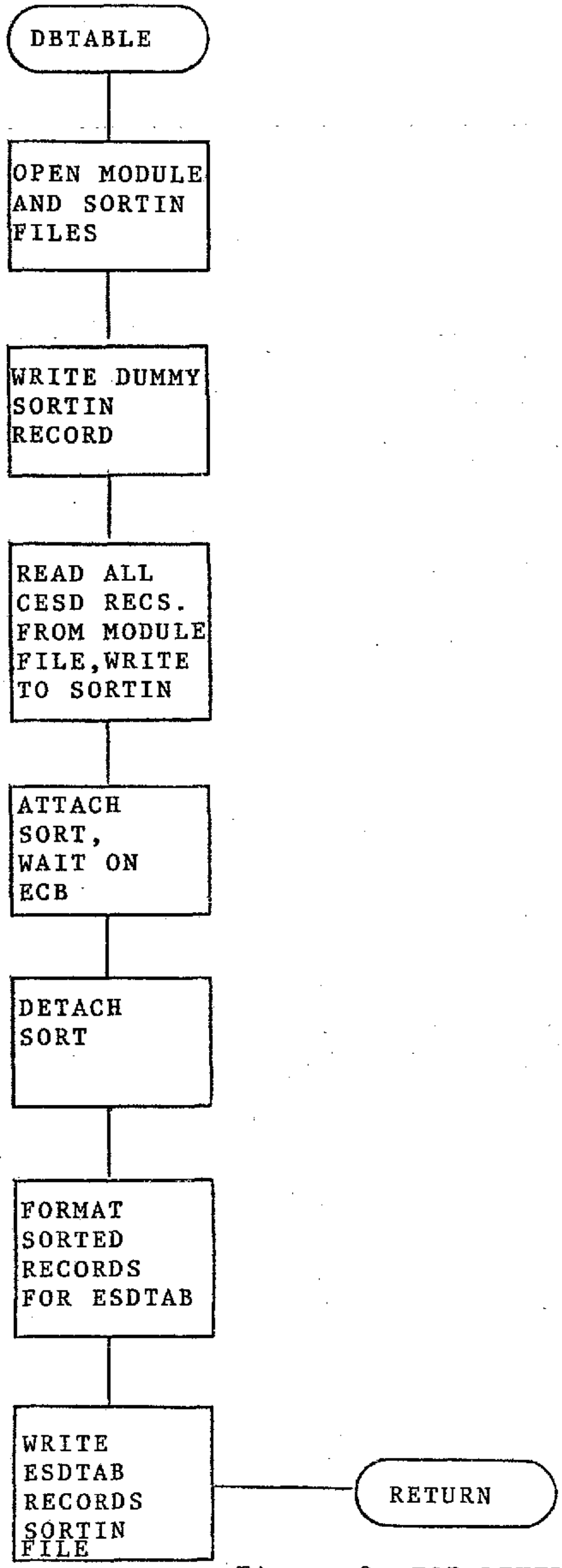


Figure 2. TOP LEVEL FLOWCHART

TOPIC C.3 - MODULE TABLE UTILITY

A. MODULE NAME:

Program-ID - NDBMTAB
Module-ID - DBMTAB

B. ANALYST

William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

This module runs stand-alone to process records from the ESDTAB data set (created by DBTABLE) and generate records for the MODTAB data set. The MODTAB records are then used during the initialization of the NASIS monitor.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

See Figure 1.

2. INPUT DATA SETS

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

SYSIN - This file is used to convey the number of regions the MODTAB data set is to have addresses and segment numbers for per transient (non-segment 1) module. If this file is omitted, three regions are assumed. This file should contain one card with the syntax

REGIONS = i

where i is the number of regions 1, 2, or 3. The keyword can start in any column.

c. Input Files

This module reads the ESDTAB data set created by the DBTABLE module; it expects a DDNAME of ESDTAB.

d. On-Line Terminal Entries

Non Applicable

3. OUTPUT DATA SETS

a. Output Files

This module creates the MCDTAB data set; it expects a DDNAME of MODTAB for this file.

The SYSPRINT file is used to provide processing information either assumed or received from the input files. Also, error diagnostics appear on this file.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

This module provides a copy of the information written to MODTAB on the SYSPRINT file. For each module processed, the time limit, region offsets, and segment members per region are written in tabular form. All values are decimal.

d. Punched Card Output Files

Not Applicable

4. REFERENCED TABLES

Not Applicable

E. PROCESSING REQUIREMENTS

1. TOP LEVEL FLOWCHART

See Figure 2.

2. NARRATIVE

Upon entry this program attempts to open the SYSIN file. If the open is successful the first card is read in and interpreted for the 'REGIONS' keyword value. If the value is not greater than one or not less than four or SYSIN could not be opened, a default value of three regions is assumed.

The first ESDTAB record is read. This record contains the number of remaining ESDTAB records there are to read.

All the ESDTAB records are read. Only the CSECT external symbols are checked and only those with a blank, '2', or '3' as the eigth character. This test provides all non-segment-one csects; i.e. transient csects. Having found one, the program saves the ESDTAB record.

When all ESDTAB records have been read, then the MODTAB file is opened and a dummy fixed record is written.

All the saved ESDTAB records are sorted so that the csects with a blank eighth character are in alphabetical order followed (each) by their '2' and '3' suffixed counterparts. For example,

DBAAAA
DBAAAA2
DBAAAA3
DBCCCC
DBCCCC2
DBCCCC3
etc.

After being ordered in this fashion, the data for each module and its '2' and '3' corresponding csects are combined into one MODTAB record and written out.

Concurrently, the same MODTAB information is written to the SYSPRINT file in tabular form.

The MODTAB file is closed and reopened. The first record is read and filled with MODTAB number of records and number of regions used; the first record is rewritten and the file closed.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in IBM PL/I language.

2. Suggestions and Techniques

Not Applicable

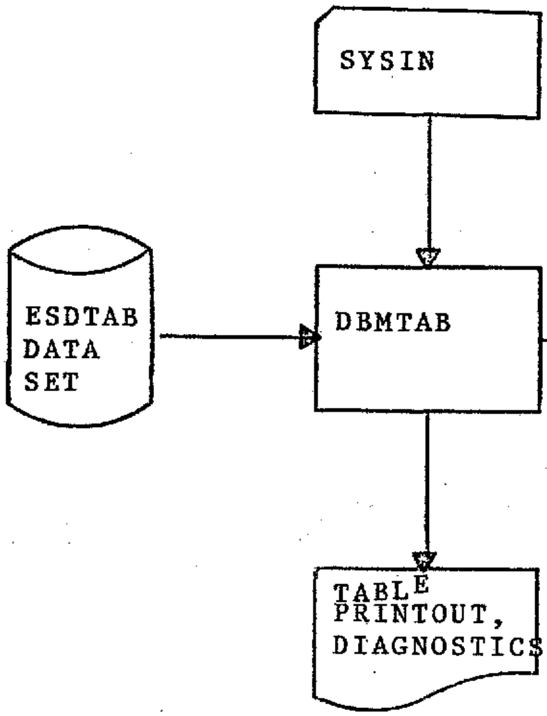


Figure 1. I/O BLOCK DIAGRAM

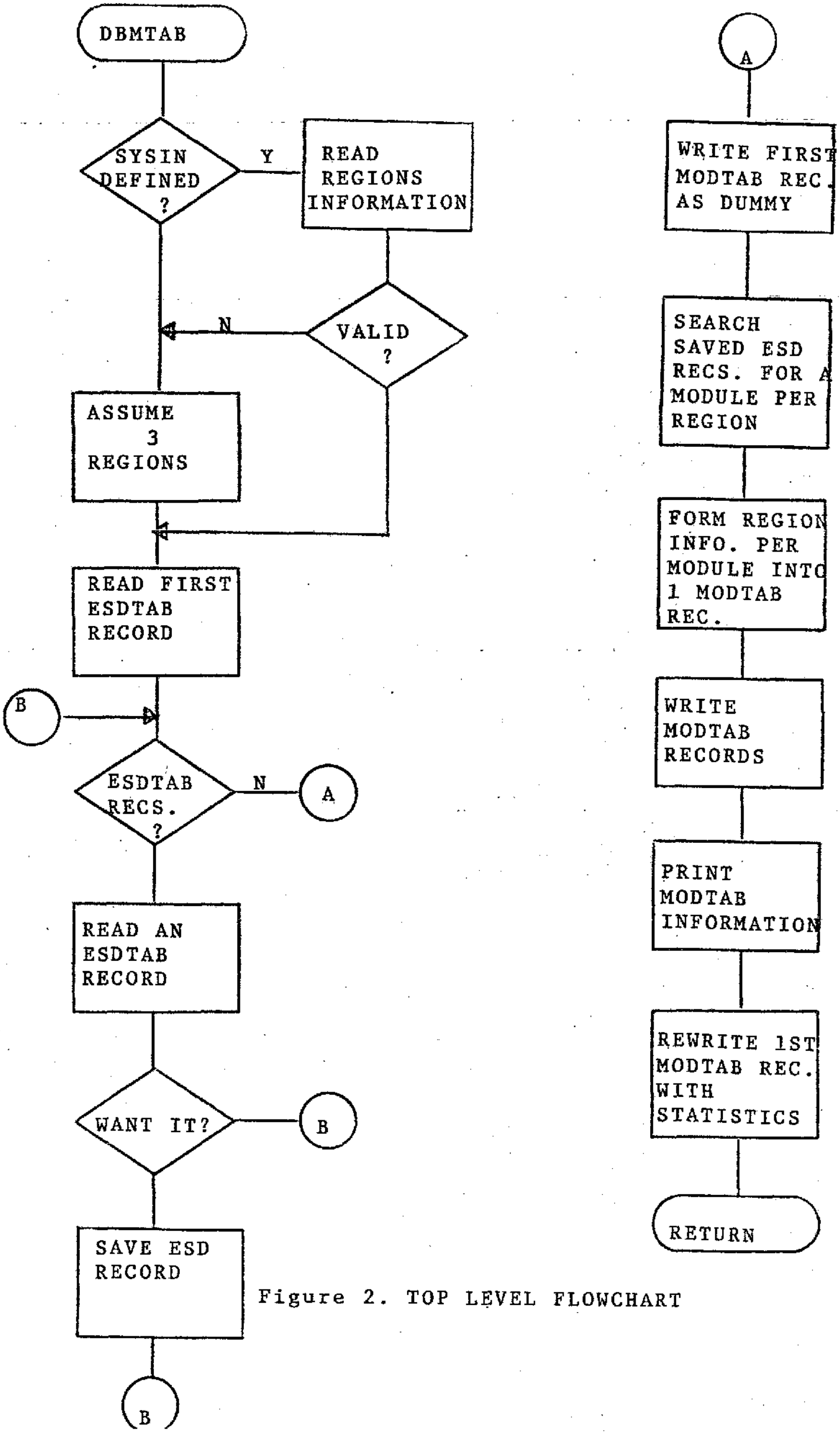


Figure 2. TOP LEVEL FLOWCHART

TOPIC C.4 - SET FILE INITIALIZATION

A. MODULE NAME:

Program-ID - NDBSETI
Module-ID - DBSETI

B. ANALYST

Tom C. Moser,
Neoterics, Inc.

C. MODULE FUNCTION

This module runs stand-alone to attach the SETINIT module to pre-format the Set File; this module also creates the Sets Information File.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   Not Applicable

3. Output Data Sets

   a. Output Files

      This module creates the Sets Information File.

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Printouts

      Not Applicable

   d. Punched Card Output Files

      Not Applicable

4. Reference Tables

   Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

This module attaches the SETINIT module to pre-format the Set File. The attached sub-task will abend with a completion code of D37 or B37 signifying the end of the allocated space was encountered. Concurrently, the SETINIT module updates the VOLTBL table. When the attached sub-task finishes, DBSETI opens and writes the data in the VOLTBL table to the Set Information file. After closing this file, the module returns.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in IBM OS/360 Assembler language.

2. Suggestions and Techniques

Not Applicable

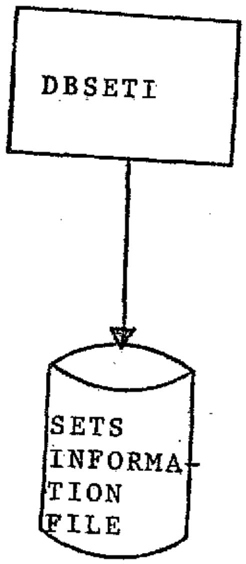


Figure 1. I/O BLOCK DIAGRAM

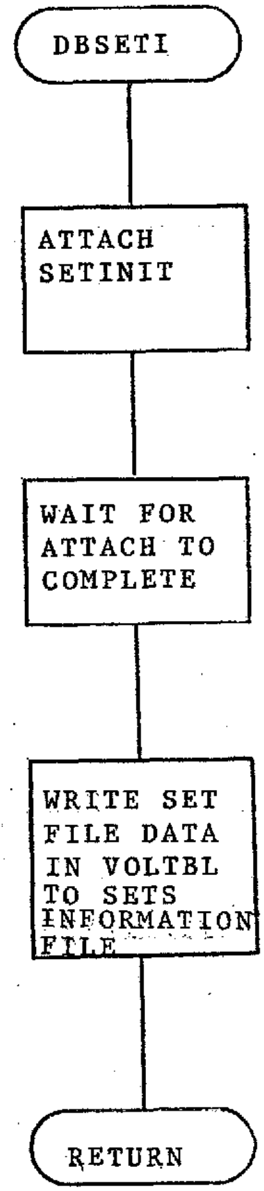


Figure 2. TOP LEVEL FLOWCHART

TOPIC C.5 - SET FILE PRE-FORMATTING

A. MODULE NAME

Program-ID: NSETINIT
Module-ID: SETINIT

B. ANALYST

Tom C. Moser,
Neoterics, Inc.

C. MODULE FUNCTION

This module is attached by DBSETI to pre-format the Set File. SETINIT writes fixed-length records to a one or multi-volume BDAM file until it runs out of space, abending with a D37 or B37 completion code. While the BDAM file is being pre-formatted, the VOLTBL table is kept current with the Set File data. Control is returned to DBSETI.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   Not Applicable

3. Output Data Sets

   This module writes fixed records to pre-format the Set File.

4. Reference Tables

   Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   See MODULE FUNCTION (C).

F. CODING SPECIFICATIONS

1. Source Language

   This module is written in IBM OS/360 Assembler language.

2. Suggestions and Techniques

   Not Applicable

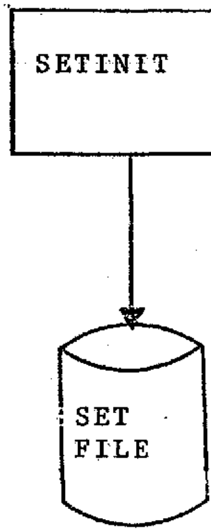
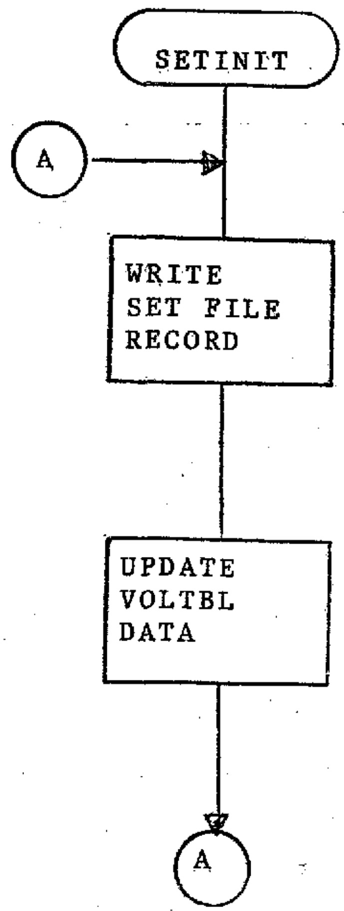


Figure 1. I/O BLOCK DIAGRAM



NOTE: MODULE WILL ABEND WHEN SET FILE IS FULL

Figure 2. TOP LEVEL FLOWCHART

TOPIC D.1 - MAINTENANCE, FREE-FORM PARAMETER PARSER

A. MODULE NAME

Program-ID - NFPARM
Module-ID - FPARM

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This routine reads in a card input file and parses program parameters that are entered on cards with Keywords. The Keywords are saved in a controlled table with their parameter values.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

   See Figure 1.

2. Input Data Sets

   a. Parameter Cards

      Not applicable

   b. Punched Card Input Files.

      The user enters parameters freeform on the sepin card input.

   c. Input Files.

      Not Applicable.

   d. On-line terminal entries

      Not Applicable

3. Output Data Sets

   Not Applicable

4. Reference Tables

   Keywords with their parameter values are saved in a controlled structure.

E. PROCESSING REQUIREMENTS

1. Top level Flowchart

See Figure 2

2. Narrative

a. Initialization

Initialize controlled table of keywords and parameters to null.

b. Read Card

Read card and strip off leading and trailing blanks. Save string in work area.

c. Parse for Equal Sign

Search for equal sign if no equal sign, print diagnostic and terminate. Save keyword in keyword_Table. IF left paren, go to section d. Go to section e.

d. Multi_comma_search

Loop through elements and save parameters in table.

e. Single_Field

Search for a comma. Save parameter in table. Go to section b if no comma. Go to section c.

When no more input cards, return to caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is coded in IBM PI/I language.

2. Suggestions and techniques

Not Applicable

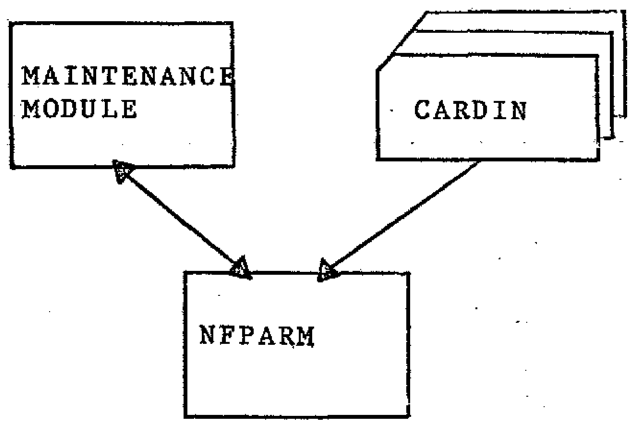


Figure 1. I/O BLOCK DIAGRAM

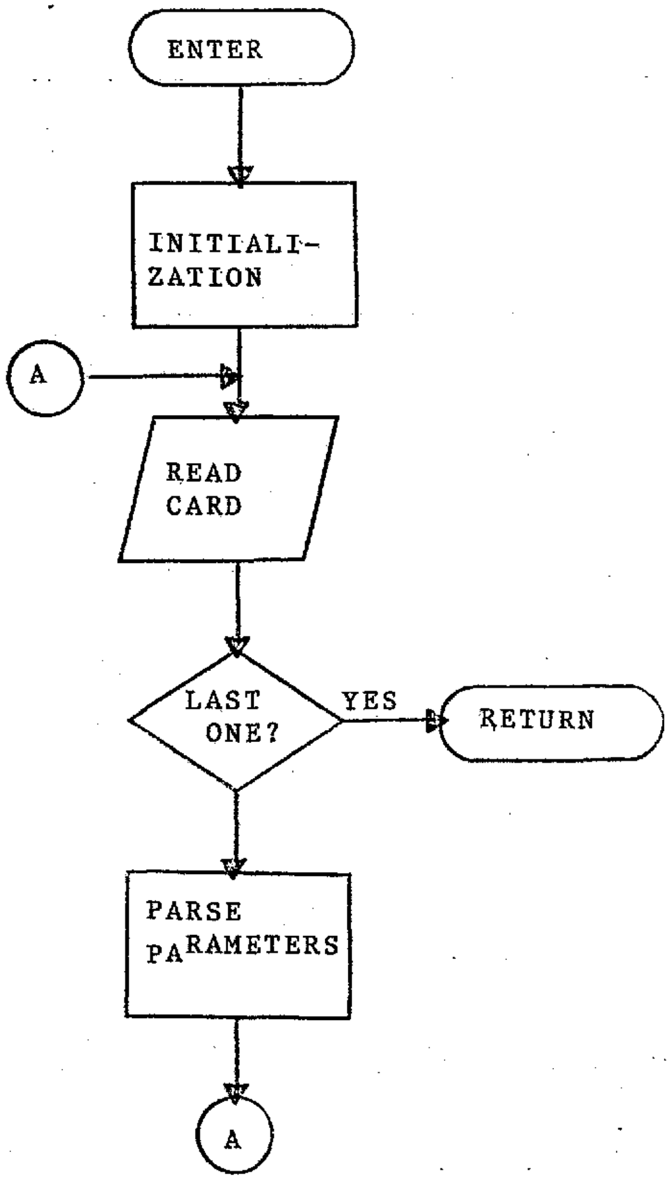


Figure 2. TOP LEVEL FLOWCHART

TOPIC D.2 - MAINTENANCE, PRINT FILE ROUTINE

A. MODULE NAME

Program-ID - NPRTFILE
Module-ID - PRTFILE

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This routine prints the print line from an external controlled structure to the sysout device which is usually the class A printer for an OS system.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card input files.

Not Applicable

c. Input Files

Not Applicable

d. On-line terminal entries

Not Applicable

3. Output Data Sets

PRTOUT is the only output dataset. This is a CLASS A print output file.

4. Reference Tables

External PRT structure is used to get the print line. First byte is the printer control character. Next 132 bytes are the print line.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2.

2. Narrative

Check for valid print control character. IF not valid, set print control character to a blank. (single space). Construct print work area from external structure. Write the print line. Return to caller.

F. CODING SPECIFICATIONS

1. Source Language

The IBM PL/I language is employed.

2. Suggestions and techniques

Not Applicable.

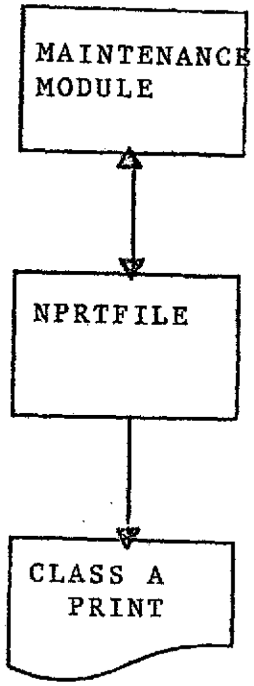


Figure 1, I/O BLOCK DIAGRAM

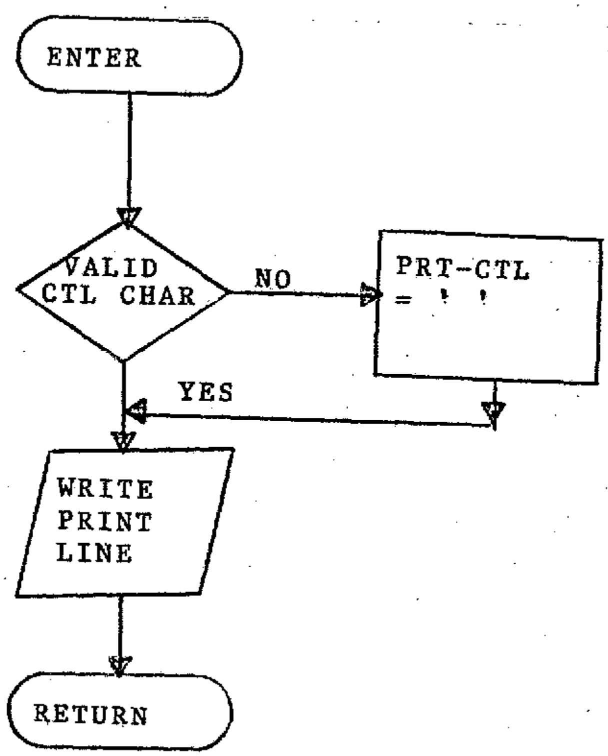


Figure 2. TOP LEVEL FLOWCHART

TOPIC D.3 - MAINTENANCE MAINLINE

A. MODULE NAME

Maintenance Mainline
Program-ID - NDBMNTN
Module-ID - DBMNTN

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The Maintenance Mainline program is an independent module which carries out any actual changes necessary to correct, update, or expand the file. The specific changes, which can be additions, deletions, or replacements, are accepted by Maintenance in the form of transactions. The transactions are kept on a data base named 'TRNSCT' and are created and maintained by the CORRECT command.

The transactions can be applied to the data base on a record, field, or element basis. Those transactions which are successfully applied to the data base are deleted. Therefore, after the successful completion of a maintenance run, the only transactions remaining on the 'TRNSCT' data base are those which need correcting. The Maintenance Mainline acquires the necessary statistics while executing and causes the 'STATIC' data base to be updated (via a call to NDBUPDST). The Maintenance Mainline is run only in batch mode. The restart capability of the maintenance run is inherant because of the deleting transactions as they are applied.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Filename must be entered as 'PARM' on the program execute card.

b. Punched Card Input Files

Not Applicable

c. Input Files

The maintenance program requires all of the files which make up a data base as input to the module.

A particular execution of maintenance may require any or all of the individual data files, depending upon the makeup of the transactions. Whereas the files in a data base are the source of the old or current data for maintenance, the transaction data base (TRNSCT) is the source of the new or replacement data (i.e., the changes). The complete description of the transaction queue is found in the dataset specifications. The transaction data base (TRNSCT) contains information concerning the data base, file, record, field and element to be maintained, as well as the type of maintenance and the new data.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The entire files of a data base may be used as output files for maintenance. As in the case where the files of a data base are used for input, the individual data files are output files only if specific transactions require them.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Since DBPL/I is used extensively in this module, the various combinations of DBPAC errors should be handled properly. These are in an array to determine program processing after error occurs.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. NDBMNTN(DBMNTN-entry point)

The Maintenance Mainline program is an maintenance module which carries out changes to the files comprising a data base. The program receives directives to modify a data base file or files from the maintenance transaction data base (TRNSCT).

b. Initialization

The key fields descriptor is read and upon finding it, the key field's length is saved.

If any errors are incurred while reading the descriptor file, the proper message is emitted and the run terminated. If not, then the use of the descriptor file is at an end and it is closed.

It is now time to initialize the transaction file by opening it, positioning it and making the first record to be processed available.

An error on opening of the transaction data base could mean that there is no data on the 'TRNSCT' data base, or that the 'TRNSCT' data base is already opened for update or output. In either case, appropriate error messages are issued and the run is terminated.

To position the data base (after opening), we do a Read by Key NOLOCK. The key we create consists of the data base name concatenated with the owners-ID contatenated with all bits off. This should represent a low key value. This yields either a successful read or a DBPAC error of 108. We expect the error to occur. Then a sequential read is performed

and we obtain the first transaction to be processed. Before continuing a get field is executed on the key and its contents are checked. If the key does not represent the proper data base name, owner-ID combination an error message is emitted and the run is terminated.

Otherwise, we are prepared for the final stage of initialization.

The regular transaction data base (TRNSCT) routine is set, the data base which is being updated has its error routine set and it is opened for direct update or sequential output.

The initialization process is complete.

c. Delete the Transaction

If the transaction is successful, it is now deleted from the TRNSCT data base.

d. Read Transaction

The transaction file is a data base which consists of only an anchor file and no associated or inverted files. The transactions are read sequentially. The Data Base Executive performs all of the necessary I/O operations. After a transaction record is located by the Data Base Executive, GETS are executed on all of the desirable fields. These fields are disseminated to various work areas. Then, checking is performed based on the presence and/or absence of data. The validation of this data is based upon the following:

See Figure 3

If there is an error detected during this initial processing, then the transaction is in error.

e. E.O.D. (end-of-data)

The end-of-data is only detected on the transaction data base. When this condition is detected, all the files are closed, appropriate messages are issued and the processing continues at the reset the

switches, section (g).

f. Reset the Switches

This section of code is executed antecedent to the occurence of an end-of-data condition. The function required at this point of time is a reseting of the 'DATA' switch on each of the descriptor regions. The files of the data base are manipulated to detect the existence or non-existence of data and the 'DATA' switches of the corresponding files are set accordingly.

g. DEL_RTN: Delete field routine.

This routine uses the #FIELD function to reput all the elements in the field to null. If it is the key field, then the entire record is deleted.

h. ADD_RTN: Add routine.

This is the add record and add element routine. If the field name is the key field then this name is stored to indicate to the maintenance routine that a new record is to be added to the file. If the field is not the key field, then a test is made to see if the transaction key is already present. If not, then the key is compared to the stored key from the last add transaction with a key field. If they do not match, an error has occurred and is flagged; otherwise, a record is created with the stored key. The new element is then put to the record. Control is passed back to section (e) on completion of this transaction.

NOTE: If subfile key is present in transaction then subfile record is obtained. If SUBCTL field is present then new subfile record is located.

i. CHG_RTN: Change Routine.

If no start or end field, given element is replaced. Using the key passed in the transaction record, the appropriate record is read in from the data base. The field is obtained, by name, as indicated in the transaction. At that point, the value of the returned field element is compared to the

'old' data element in the transaction. If no match is made, a test is made as to whether the returned element is null, thereby signifying the end of the field. If that is the case, then an error has occurred and is indicated. If the null element was not detected, then the next element is obtained and the process repeated. If a match does occur, then the 'new' data element from the transaction record is reput to the record. If the 'new' data element is null, then the element is deleted. Continue processing with section (e). If a start and end field is present then a field context operation is performed.

The maintenance program can carry out changes to portions of large fields without the entire field cn the transaction entry of record. To begin, the record is read into a large enough area to hold the maximum record using the key provided in the transaction. The field in question will then be obtained and an iterative process is applied wherein the 'old' data value is compared sequentially across the field from the starting location to the ending location. Whenever a match is found, the 'new' data value is used to replace the 'old' in the field and a count is kept of the number of replacements. When the end of the search range is reached, the count is tested. If no matches were made, then that error is recorded. The processing will continue with section (e).

F. CODING SPECIFICATIONS

1. Source Language

As much as possible of the Maintenance Module is coded in the IBM programming language PL/I. The input and output coding for access to files in a data base is handled through an extension to that language, known as DBPL/I.

2. Suggestions and Techniques

a. Much of the verification of correct access to files in a data base is handled within the DBPAC routines. Full advantage of these features was taken for all I/O processing.

b. In the preceding narrative, not every instance of the need for an informative message was indicated. During implementation, all appropriate messages are included to increase the understanding of the user.

c. While not noted in the narrative, it is necessary to test the return codes from every input and output operation. In those cases where errors occur, messages are written out and the task terminated unless a correction can be applied, in which case the processing can then continue.

d. Whenever it becomes necessary to terminate the maintenance routine at any point, it is desirable to make every attempt to restore the data base to a normal condition. In most cases, this action involves resetting control switches found in the header records of the descriptor file. This action makes possible subsequent processing on the data base which might correct the original problem and also allows continued retrieval from all usable portions of the data base.

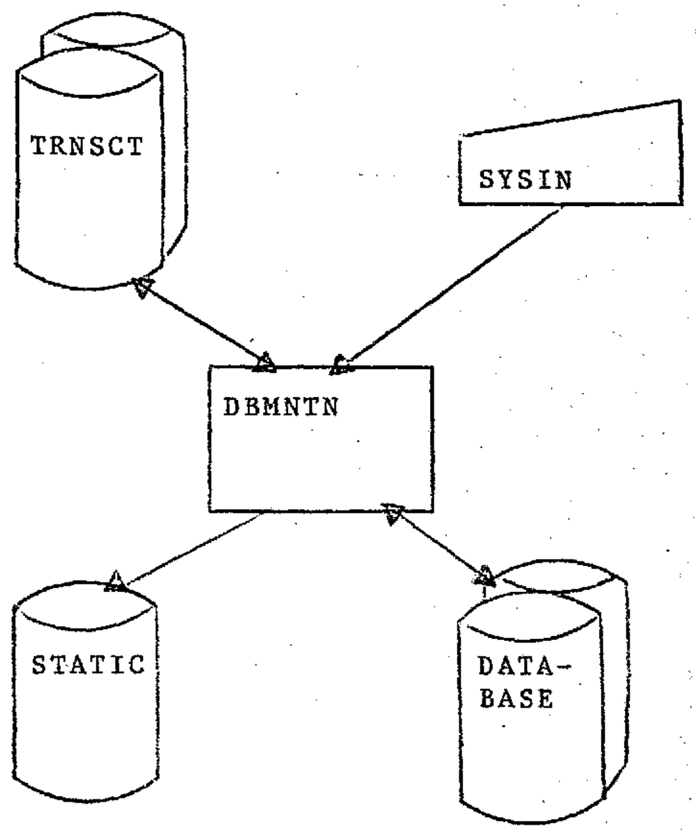


Figure 1. I/I BLOCK DIAGRAM

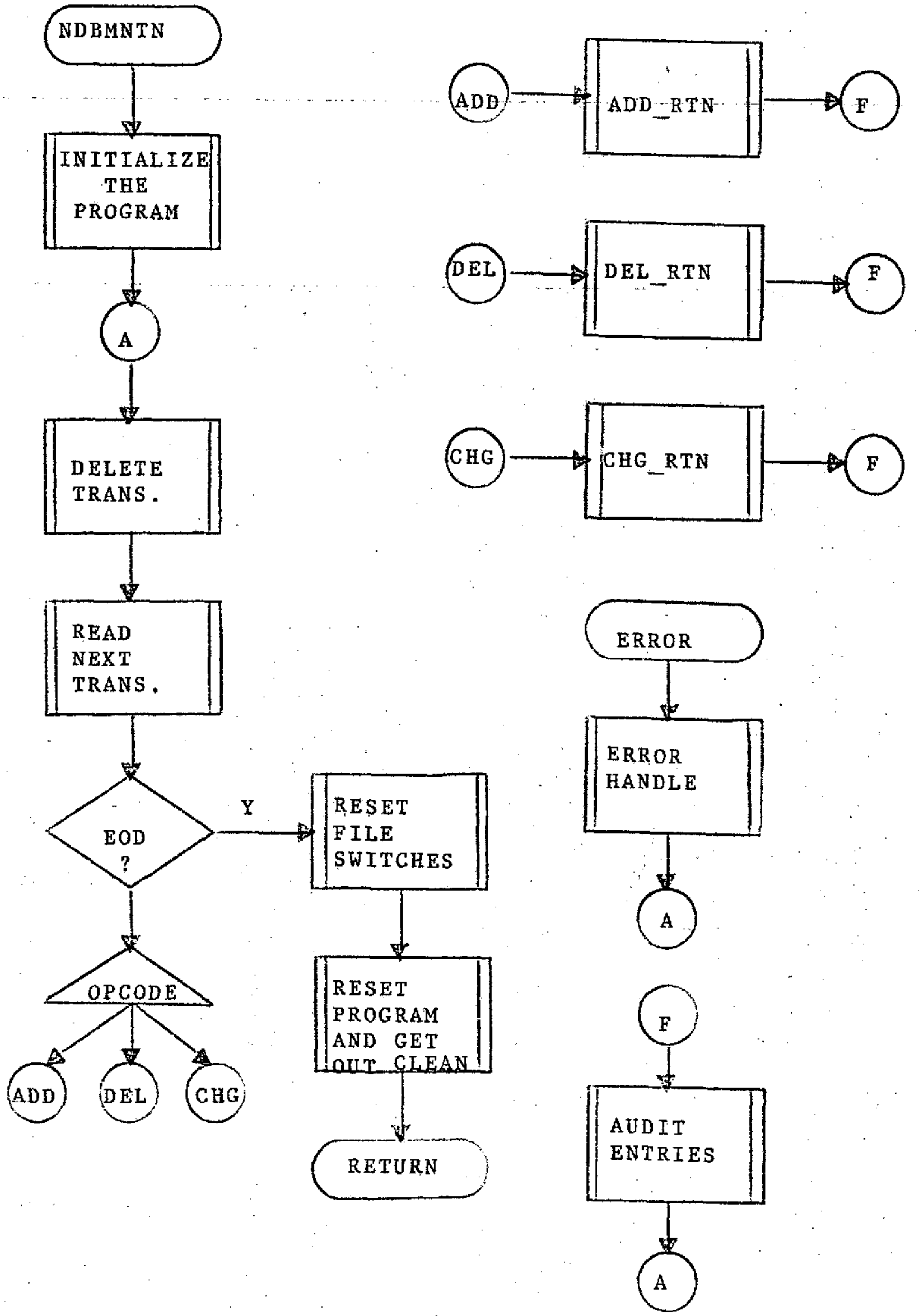


Figure 2. TOP LEVEL FLOWCHART

√ : REQUIRED PARAMETER

X : NOT REQUIRED

	ADD	DEL	CHG
KEY	√	√	√
NASISID	√	√	√
OPCODE	√	√	√
FIELD	X	√	√
START AND END	X	X	X
OLDDATA	X	√X	√
NEWDATA	√X	X	√
SUBKEY	√X	√X	√X
CTLFLD	√X	√X	X

Figure 3. PARAMETER TABLE

TOPIC D.4 - MAINTENANCE LOAD/CREATE

A. MODULE NAME

Load/Create
Program-ID - NDBLOAD
Module-ID - DBLOAD

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This module provides a generalized file loading capability for NASIS.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Parameters are keyworded and freeform on sysin card input.

b. Punched Card Input Files

Not Applicable

c. Input Files

1. The primary input file is the data set containing the records to be loaded to the data base. This file must be indexed sequencial, with the keys having the same format and valu as thats that of the final data base.

2. The only other input file is the descriptor data set for the data base being loaded.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

   a. Output Files

      1. The primary output file is the data base which is being loaded. It must have its descriptors fully defined beforehand, but all other functions will be handled by NDBLOAD.

      2. The other output file is the error file, on which is written exact duplicates of any input records that cannot be successfully loaded.

      3. A print file for diagnostics and program status information is printed on Class A output.

   b. On-line Terminal Displays

      Not Applicable.

   c. Formatted Print-outs

      Not Applicable

   d. Punched Card Output Files

      Not Applicable

4. Reference Tables

   The module contains a table of error switches which control the action to be taken for each possible DBPAC error; abend, skip record, or skip field.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   a. Upon entry the program establishes interrupt handling routines which will terminate if any PL/I errors occur, or display statistical data.

   b. The program next reads in and parses the input parameters applying defaults for any

parameters that are not entered.

c. The next step is to open the descriptors for the file specified. The file header switches are reset in case the system crashed. The index file headers are read, and if field not indicated to invert, the loadable switch is turned off.

d. The next function performed is the definition and opening of the input file, the error file and the data base itself. At this point, the program checks the user's mode parameter, and if it is restart passes control to section (g) before continuing.

e. Finally, the program is ready to process data. It reads an input record, passes the record to the user written exit routine for separation into its component fields. Upon return from the exit routine, the program tests the status bits, and if set properly, begins writing the input data to the data base, field by field. If any errors are detected, and appropriate diagnostic is written to the user and the action indicated by that error's code in the ERROR_CODE table is taken. The options are to abend the program, to skip the remainder of the record, or to skip the field. When the field has been completely processed, the routine continues with the next input record, until the data is exhausted.

f. When all of the data has been processed or when a terminal error has been detected, statistical counts are written on the user's terminal along with a termination message. All of the files are closed, and the status bits of the descriptor header records of each of the component files of the data base are posted to indicate whether date exists on the file or not. The program then terminates.

g. If the user specified a restart, the program retrieves the last record written to the data base. It then accesses the next record to be written from the input data set. When the operation is complete, processing continues with section (e).

F. CODING SPECIFICATIONS

1. Source Language

   This module should be written in the PL/I Language.

2. Suggestions and Techniques

   a. Because of the function of this module, extreme care should be taken to code it as efficiently yet as indestructibly as possible.

   b. Any place in the program where there is any remote possibility of an error, there should be a meaningful diagnostic.

   c. The ERROR_CODES table was designed to be used in conjunction with a label array. The digits in the table are to be converted to index values and an indexed branch taken based onthe label array.

   d. The user-written exit routine is responsible for assigning field names, field off-sets, and field lengths.

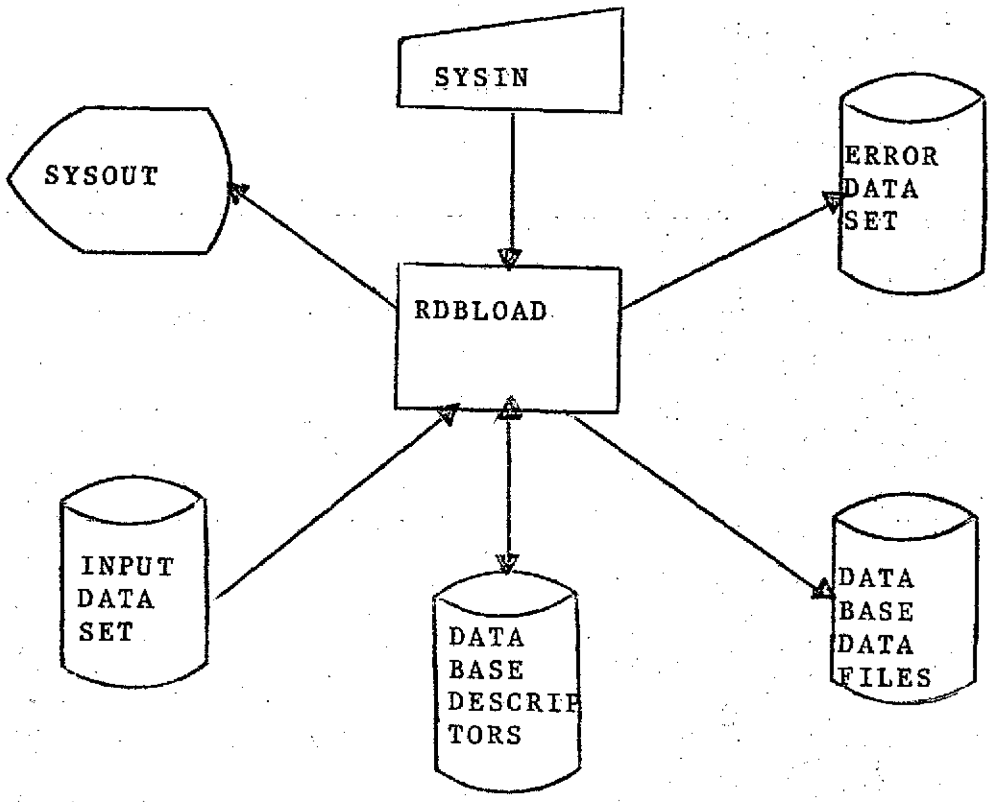


Figure 1. I/O BLOCK DIAGRAM

TOPIC D.5 FILE INVERTER

A. MODULE NAME

Maintenance - File Inverter
Program-ID - NDBIVRT1
Module-ID - DBIVRT1

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The inversion program (NDBIVRT1) is a maintenance program for data base file creation. The purpose of the program is to take data from certain fields of a data base and to post this data to an inverted index file. This operation can be done automatically by DBPAC during a normal file loading operation, but it is very time consuming and could therefore jeopardize the successful completion of the load. Further, by separating this function out, in this manner, the capability of creating inverted indices after a file has been loaded and used is added to the repertoire of the NASIS system. Finally, this separation also permits the use of specialized techniques suitable specifically to this function to reduce the amount of time required for the entire process of loading and index creation.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Parameters are entered freeform with keywords on CARDIN input file.

b. Punched Card Input Files

Not Applicable

c. Input Files

1. Data Base: The primary input to the

Inversion module is the file being inverted.

2. Data Base Descriptors: The file descriptors are needed to provide information.

3. Restart file: If the program is invoked in restart mode, a restart file with the restart key is needed.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Sortin File: This file is a QSAM file with the value of the field being inverted concatenated with the file key. This file becomes the input to a DSORT utility.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Parameter Parsing routine

Call PPARM routine and save parameters in appropriate save areas.

b. Field stripping routine (step one)

If restart run, read in restart key and read this file record. If range run, read first file key to start at. Read sequentially the input file, save the internal file key. Loop through the index file suffixes. Loop through the field name table. Loop through the #FIELD for this field. Write out a sortin file record. If end of file reached, terminate. If end range key reached terminate. If number of records to process is reached, write out restart file and terminate.

F. CODING SPECIFICATIONS

1. Source Language

The Inversion program employs the IBM PL/I programming language. The special extensions of that language, called DBPL/I is utilized for all access to files in the data base.

2. Suggestions and Techniques

Not Applicable

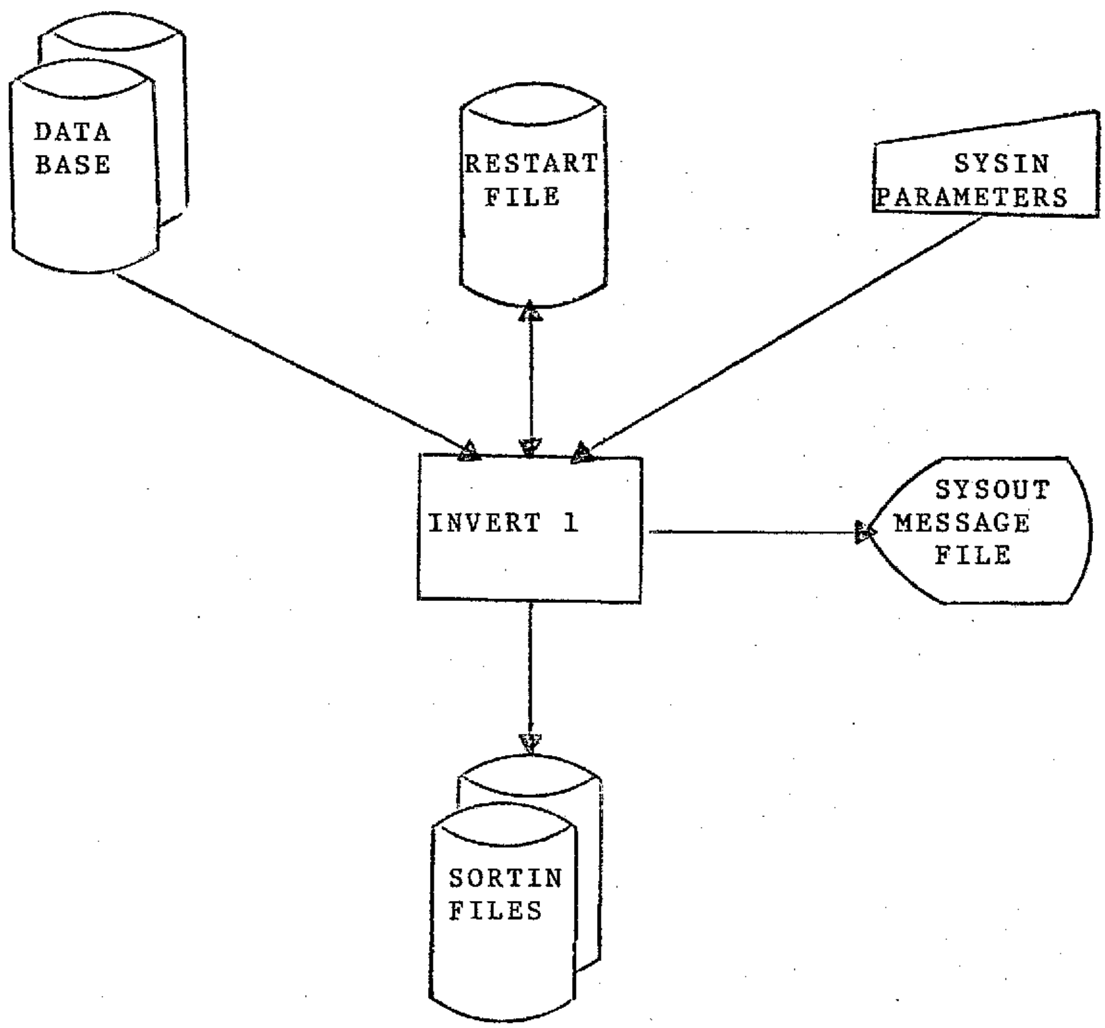


Figure 1. I/O BLOCK DIAGRAM

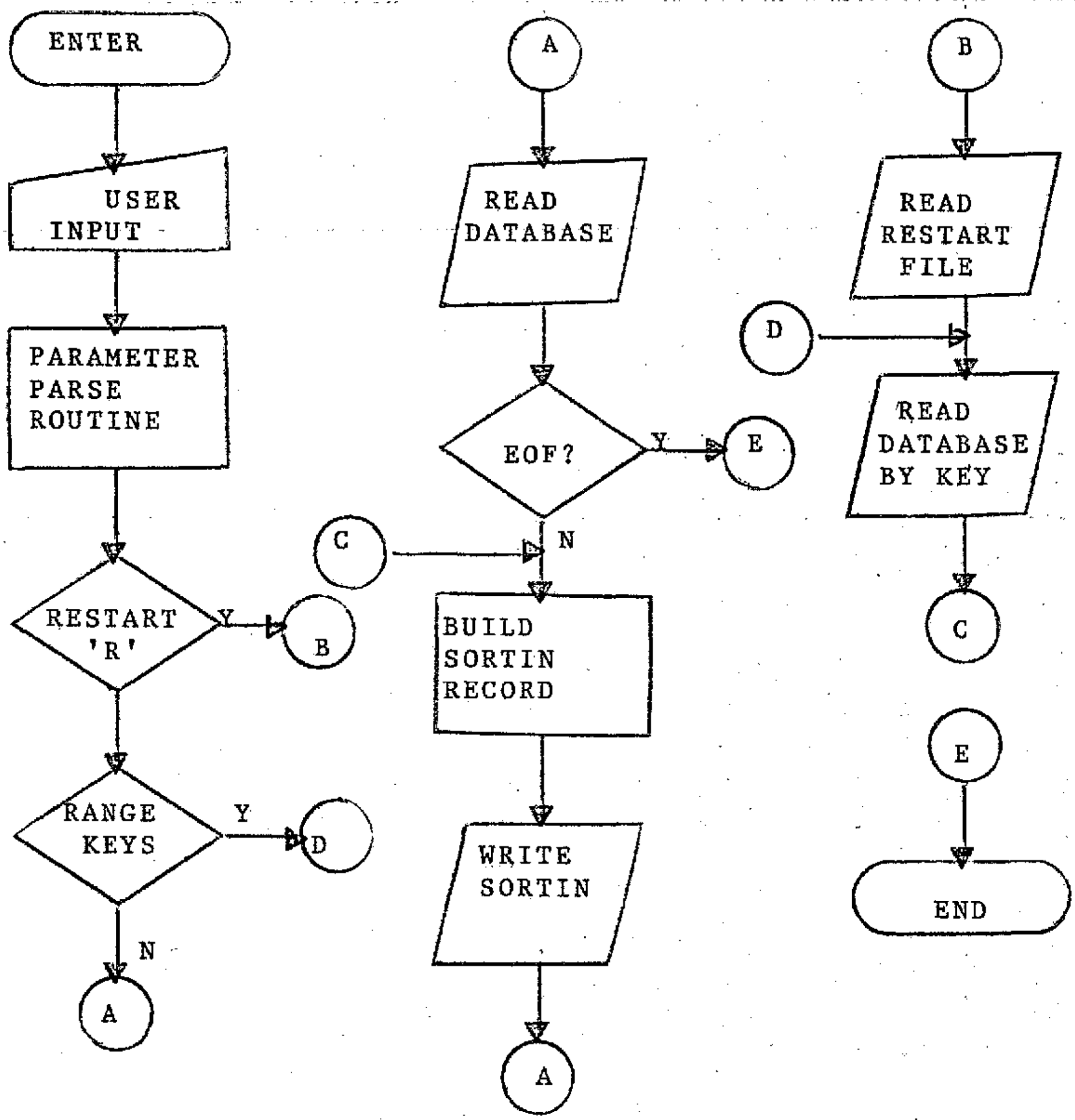


Figure 2. TOP LEVEL FLOWCHART

TOPIC D.6 - MAINTENANCE, FILE INVERTER2

A. MODULE NAME

Program-ID - NDBIVRT2
Module-ID - DBIVRT2

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This module reads in the sorted file from DBIVRT1 and creates an inverted index file for a data base.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Parameters are entered freeform with Keywords on CARDIN input file.

b. Punched Card Input Files Not Applicable

c. Input Files

1. Database Descriptors: The file are needed to provide information.

2. Sorted file from DBIVRT1. This file has field values concatenated to file key in sorted order.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

1. PIEX File: This file is the output of step three in the form on an index file with the internal field value as the key.

This file becomes the input to step four, the Translation step for external indexing.

2. Range File: This file is the output of step three if field is indexed with internal format, and is the output of step four if field is indexed with external format. Range of keys to invert must have been specified for this file to be produced. This file becomes the input to the index merge program.

3. Database Index File: This is the final index file. It is the output of step three if internal indexing, and is the output of step four if external indexing.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top level Flowchart

See Figure 2

2. Narrative

a. Parameter Parsing routine

Call FPARM routine and save parameters in appropriate save areas.

b. Get Descriptor information

Find the key length of the QISAM output file. This will be the maximum length of the fields being inverted on the same index file. IF field is indexed external, use length. IF index file is spanned, increase

by one.

c. Write QISAM File

Read input file. IF Keys are the same, concatenate file key onto list of keys. IF list has reached maximum list length, up the span character and initialize list to null. IF keys are different, write out index record. IF end of input file reached, write out last index record and check to see if external indexing. IF external indexing, proceed to next section. Display record counts for user and post the index descriptor data switch.

d. Translate Keys routine

Read input file sequencially. Search key for first blank character after first non-blank character. Use this parsed string to pars to field formatting routine. Replace internal value in key with external value. IF end of file reached, post data bit in index header descriptor record. IF more field names in table. go to translate keys routine again. Terminate the program.

F. CODING SPECIFICATIONS

1. Source Language

The IBM OS PL/I programming language is used. The special extension of that language, called DBPL/I is utilized for all access to files in the database.

2. Suggestions and Techniques

Not Applicable.

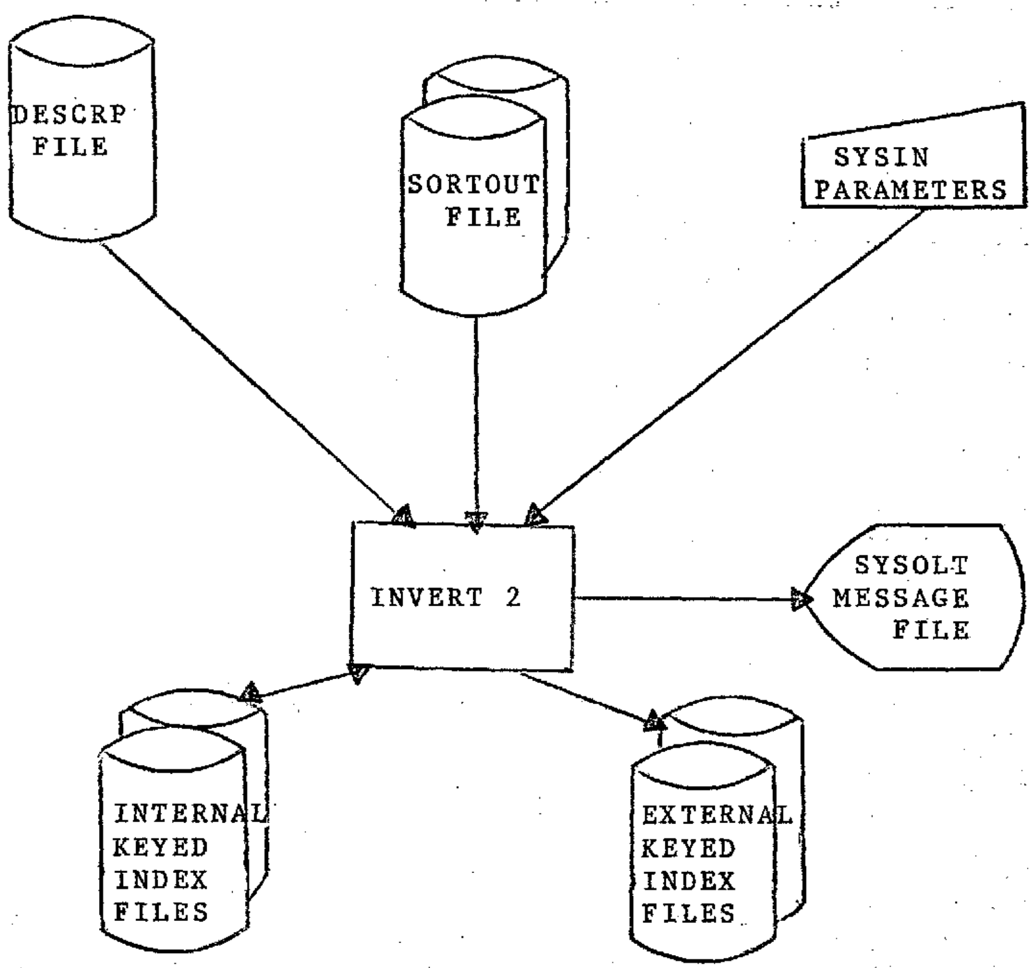


Figure 1. I/O Block Diagram

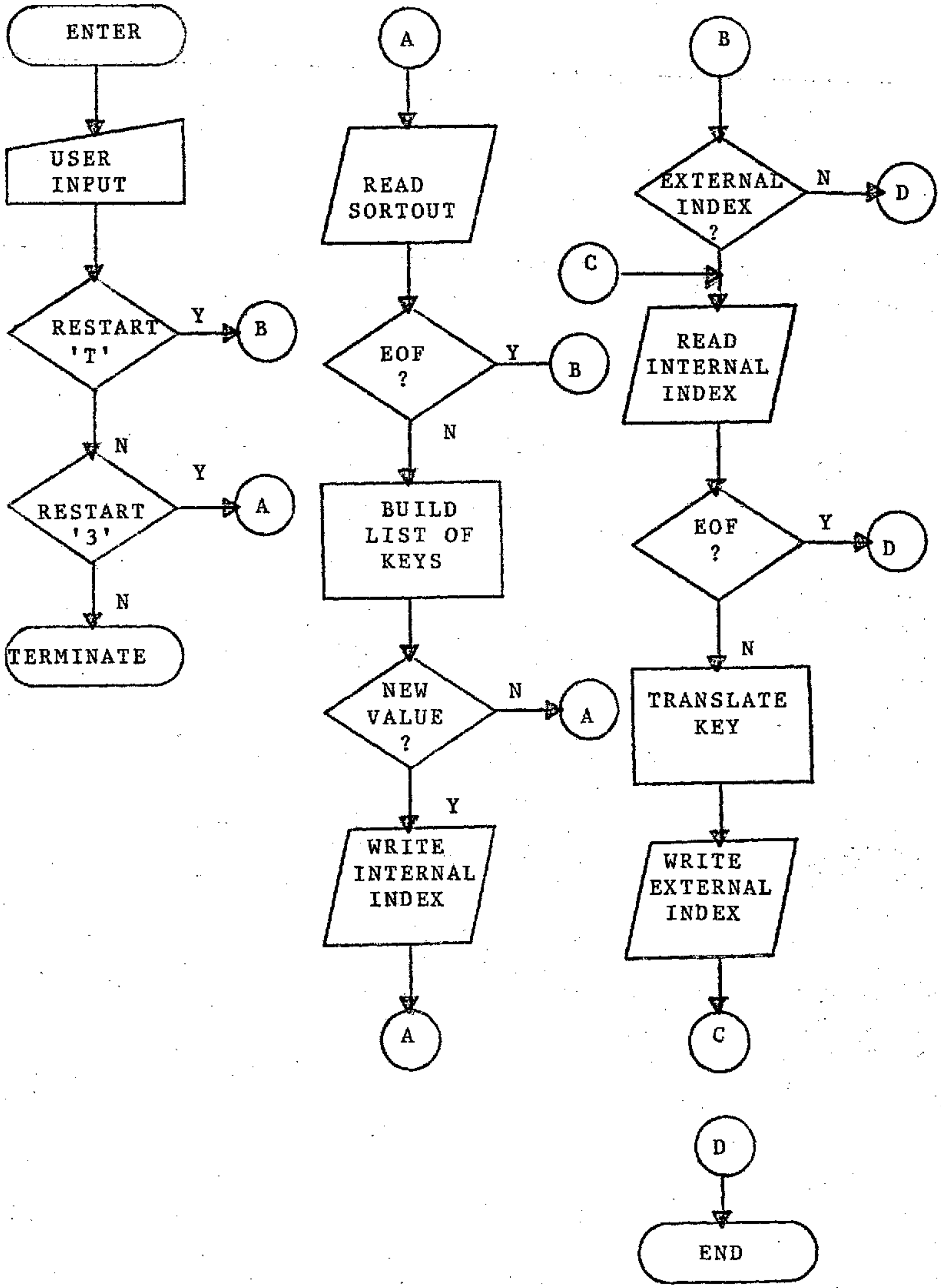


Figure 2. Top Level Flowchart

TOPIC D.7 - INDEX FILE MERGE

A. MODULE NAME

Maintenance - File Merger
Program-Id - NDBINDM
Module-Id - DBINDM

B. ANALYST

Edward McIntire
Neoterics, Inc.

C. MODULE FUNCTION

The merge module is an independent program whose function is to create an updated index file for a database. The updating of the index file can be done in place or to a new index file. This new index file will be named 'INDMRG.' ||FILE NAME||'.'||FIELD NAME' and it will have to be renamed upon completion of the merging operation. This module will also allow for the processing of duplicate records if deemed necessary.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

The merging utility is most often invoked from a terminal in conversational mode. However, it is possible to initiate the task in the non-conversational mode, just as in the case of any other os task. In batch mode, the format of the punched card input is the same as when terminal input is used to invoke the routine.

c. Input Files

1. Index File: The primary input to the merge program is the current index file that is to be updated, and the update

index file that is to be merge with the current index. The anchor descriptor file is needed to provide field informaticn.

2. Parameter File: If the user wishes to stop processing he may do so by pressing attention and responding 'y' to the prompt message. Thus a parameter file is created for input to further restarts. This will minimize the chance for user input error and insure restart at the proper key. This file consists of the last key processed on the current index file. This file is a SAM file used only in the index merge program.

d. On-Line Terminal Entries

All of the terminal entries to the merge program are in the form of responses to prompting messages from the program itself. The one exception to this rule is the initial command with its parameter to invoke the procedure. The purpose of the terminal entries are to establish file and field names, new file or inplace merge, process dups or not, firstpass or restart, and if the user wishes to quit processing or not.

3. Output Data Sets

a. Output Files

The output data set is the index file created by the merge program. This data set can take two forms:

1. The current index file updated inplace.

2. A new file created by the merging of the current index with the update index.

b. On-Line Terminal Displays

All on-line terminal displays for the merge program follow the same format. The TSPL/I facility of the system is utilized to request entries at the terminal and display progress information.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Prompting

Prompt the user for first pass. If it is not the first pass, go and read the parameter file. Prompt the user for the anchor file name, for new file creation or merge in place, for the processing of duplicates or not, and for the inverted field name. The user must enter a valid response to all of the prompts or he will be reprompted. A read sequentially of the anchor descriptors is done until a fixed field with an offset of four (4) is found. That, in fact, is the key descriptor and its length is saved. The index field is also checked for validity, if it is not a valid index, then a reprompt is initiated. Following this the index descriptors are opened and read sequentially until the index field length is obtained, and the spanned indicator is checked and it's value is saved. In all of the above cases if a critical error is encountered an error message is displayed and the program is terminated.

b. Write Parameter File

If the user deems it necessary to stop processing during the merging operation, he can press the attention button and the total records processed will be displayed. Also, a message will be displayed asking if he wishes to quit processing. When the user replys with a quit processing command the following occurs:

1. Quit switch is set.

2. Processing is continued until a clean close can be carried out.

3. Parameter file is opened and the key of the last record written is written to the parameter file.

4. Parameter file is closed.

5. Program branches to end of job routines.

c. Read Anchor Descriptors

DBPAC is utilized to read sequentially the anchor file descriptors and retrieve the field that is indexed and the anchor key length.

d. Read Index Descriptors

DBPAC is utilized to read sequentially the index file descriptors and retrieve the index key length and the spanned indicator.

e. Read Parameter File

If not the first pass, the parameter file is read to get the needed key for the restart. The restart key is used as the key and a get by key is done on the current index file. Also a read by key is done on the update file to find its starting position. From here we go to normal reads on the input files for further processing.

f. Write Index File

The writing of the index file can take two different forms.

1. Merge Inplace.

If the user decides to merge to the current index file the new records will be built in core and tabled there until a unique record is read. If a record is longer than the maximum allowable length, then create a spanned record by adding one to the spanned record character. Then rewrite any existing

records and write any new records that may have been created. When an update record does not match a current record, the update record and any with the same key are written to the current file.

2. Merge to New File.

The merge to a new file is much the same as the merge inplace. The differences are listed below:

a. An out put file is created with the same attributes as the current index file.

b. There will be no rewrites to the new file.

c. All current and update records will be written to the new file. If either file finishes first the other will be read and written until it is finished.

g. Attention Interrupts

Attention interrupt handling was discussed in section 'E', sub-section '2', Item 'B' (Write Parameter File). Any questions you might have concerning this area should be referred there.

F. CODING SPECIFICATIONS

1. Source Language

The merge program employs the IBM PL/I Programming Language. The special extensions of that language, called DBPL/I and TSPL/I, are utilized for access to file descriptors in the database and for all terminal communication, respectively. Also, the merge program employs assembler routines to handle all I/O during the execution of this program, except for the writing of the parameter file which is handled exclusively by PL/I.

2. Suggestions and Techniques

Not Applicable

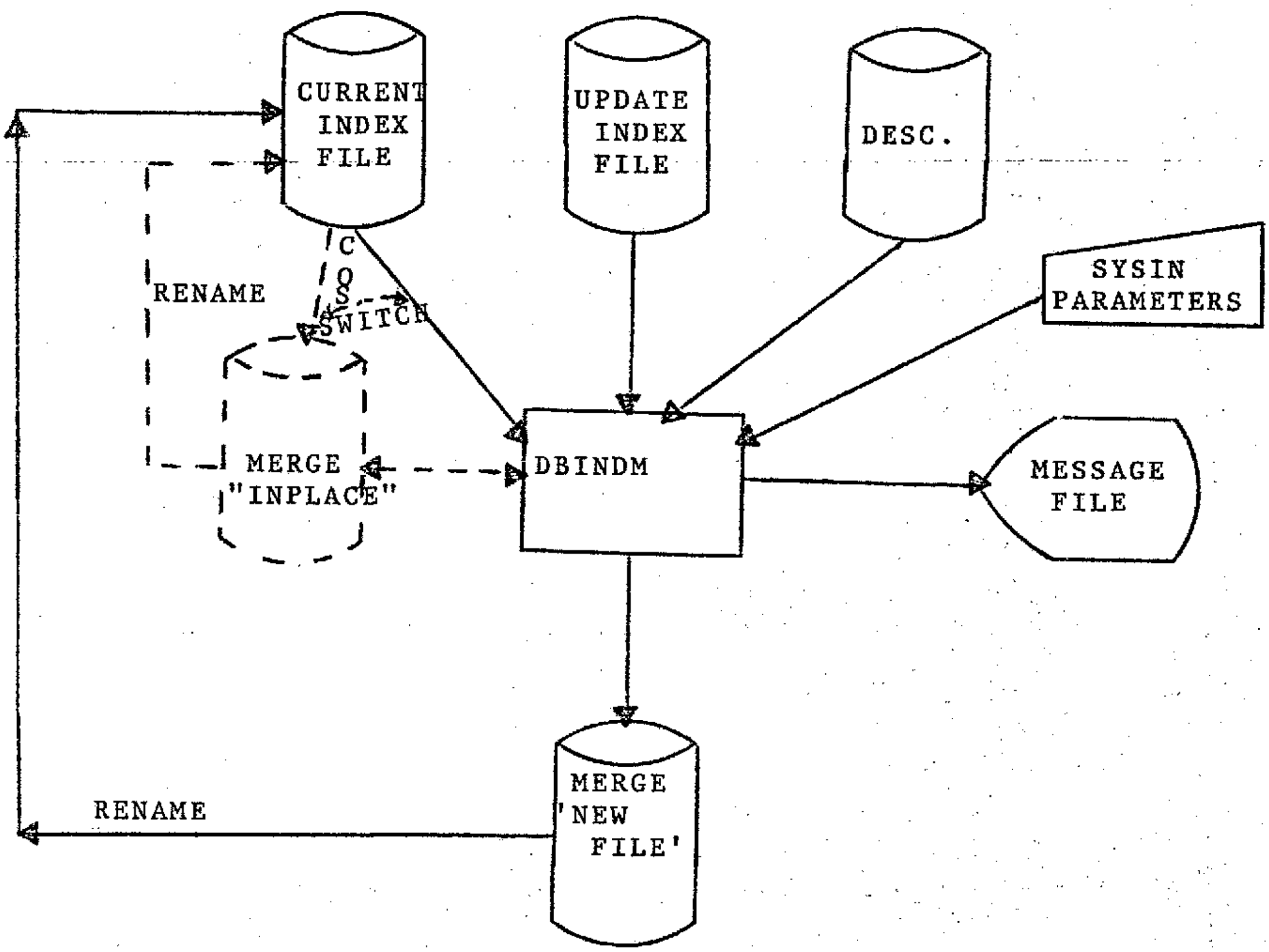


Figure 1. I/O Block Diagram

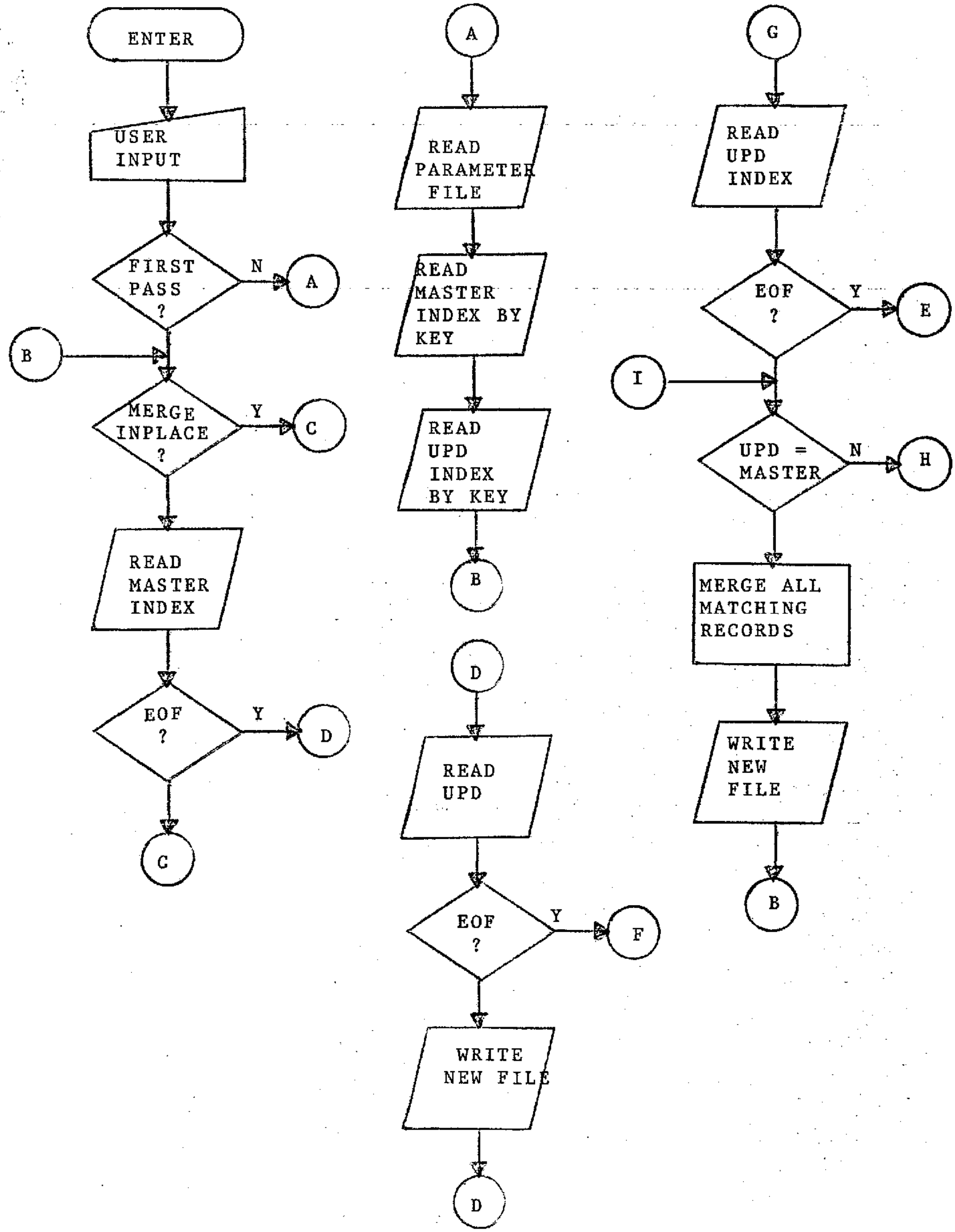


Figure 2A. Top Level Flowchart

TOPIC D.8 - DATA BASE RECORD LENGTHS

A. MODULE NAME

Program-ID: NDBRECL
Module-ID: DBRECL

B. ANALYST

William H. Petrarca,
Neoterics, Inc.

C. MODULE FUNCTION

This module executes stand-alone to read one or all files of a data base (excluding the descriptors) to determine the maximum record length within the respective file(s). When found, these maximum record lengths are posted in the respective file header descriptors as the second variable field, just past the security codes. Although the record length is posted as a variable field, it is always a full word (4-bytes) preceded by a two byte field length with a value of 6. The DBPAC module interrogates each header record for the presence of this field at every data base open. When present this value is used to allocate the size of the input buffer for the respective files. If absent the default value of 3996 is used as a maximum.

Only DB2 type descriptors are accepted by this module.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

The EXEC card PARM character string must contain the data base or data base file to sweep. The string must be of the form 'FILE = name' where name can be just the data base name, such as DB2TDB, to imply that all files of that data base are to be analyzed or the name can be the data base file name, such as DB2TDBZ, to imply only one file to analyze.

b. Punched Card Input Files

Not Applicable

c. Input Files

This program reads the particular data base descriptors along with any other data base files to be analyzed. The descriptors and the requested files to be analyzed must have DD cards provided; the DDNAMES are arbitrary.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

This module rewrites the file header descriptor records of the data base descriptors for those files analyzed.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

This module provides messages with the maximum record lengths found per file on the SYSPRINT file. Any errors encountered are also diagnosed.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry DBRECL validates the parameter

string; invalid parameters are diagnosed at SYSPRINT and the program returns. With a valid parameter, DBRECL assumes a parameter of length less than seven characters to imply an entire data-base name; greater than seven implies a particular file name.

If all the files of a data base are to be done, then the descriptors are read sequentially to determine all the file names. The names are saved in a list, and the descriptors are closed.

If only one file is tc be analyzed, then it is put into a list of one.

Then, for each file name in the current list the following is done. The file is opened and read sequentially. Each record length is compared against the going maximum. If it is greater, it becomes the going maximum. After all records have been read, the file is closed and the arrived maximum is saved in a 'length' list. The found maximum for the file is printed on SYSPRINT.

After all the files have been read, the descriptors are opened again. The header descriptors are read for each file name in the list. If a maximum record length already exists on the record, it is overlaid with the newly found maximum. If none exists, the header record length is increased by 6 and the newly found maximum is added on the end as a variable field. The header record is rewritten. The descriptor file is then closed.

For all file I/O, the ISAM assembler routines in the DBDBIO module are used. Any DD or OPEN errors are diagnosed with processing continuing with the next file in the list. Likewise, any read errors are diagnosed and the file skipped.

F. CODING SPECIFICATIONS

1. Source Language

This module is written in IBM PL/I.

2. Suggestions and Techniques

Not Applicable

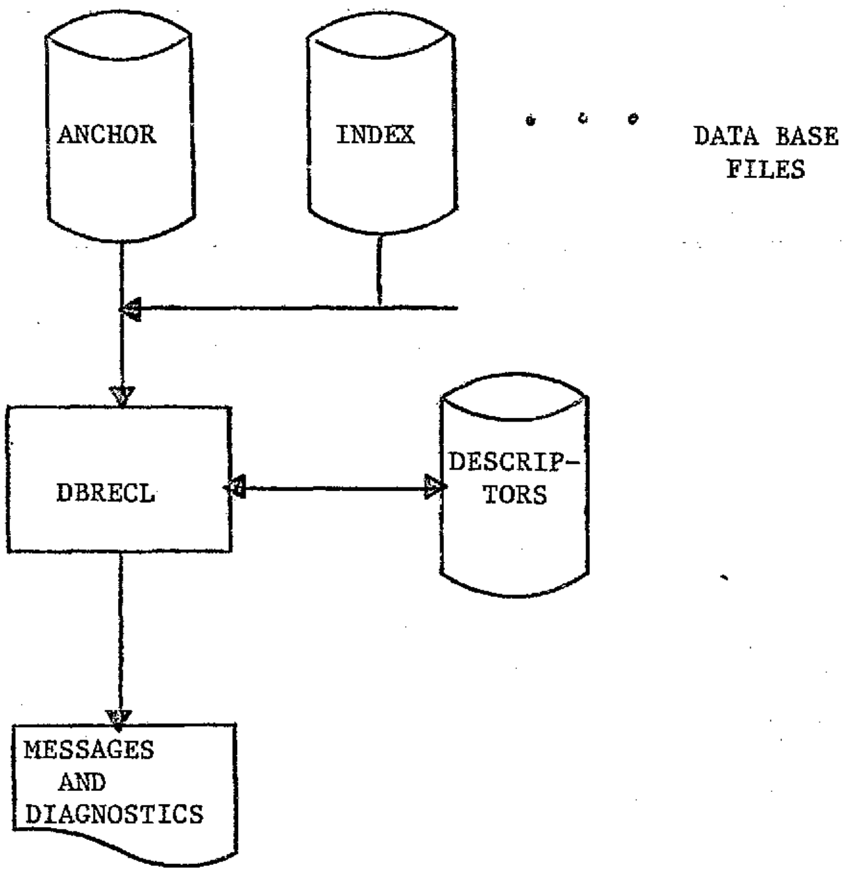
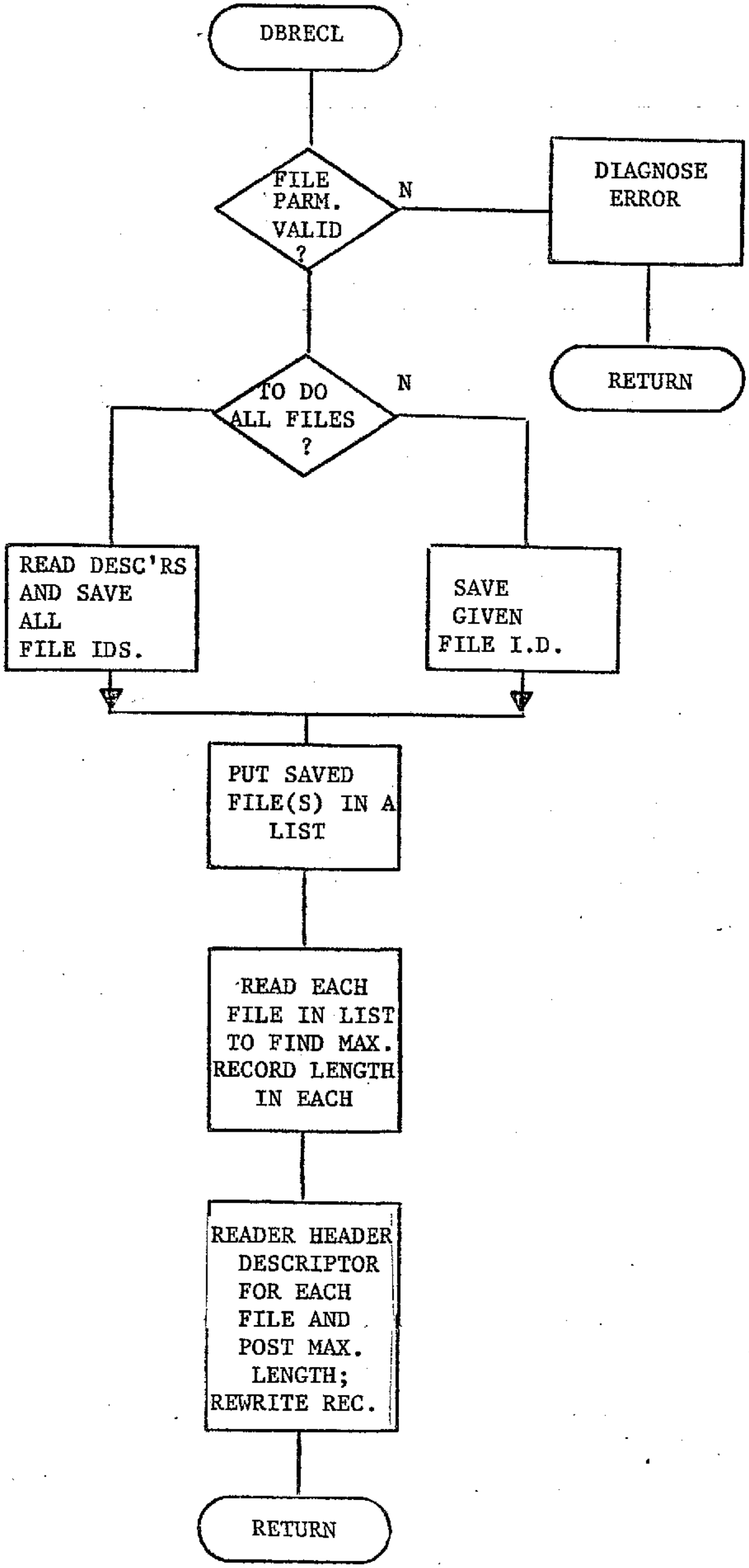


Figure 1. I/O Block Diagram


DBRECL
FILE PARM. VALID ?
N
DIAGNOSE ERROR
RETURN
TO DO ALL FILES ?
N
READ DESC'RS AND SAVE ALL FILE IDS.
SAVE GIVEN FILE I.D.
PUT SAVED FILE(S) IN A LIST
READ EACH FILE IN LIST TO FIND MAX. RECORD LENGTH IN EACH
READER HEADER DESCRIPTOR FOR EACH FILE AND POST MAX. LENGTH; REWRITE REC.
RETURN

TOPIC D.9 - DESCRIPTOR EDITOR - ADD - CHANGE COMMANDS

A. MODULE NAME

Program-ID - NDBEDAC
Module-Name - DBEDAC

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

Those commands allow the user to create and modify field descriptors.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Print-Outs

      Not Applicable

4. Reference Tables

   The following external tables are referenced by NDBEDAC:

   1. FIELD
   2. FLD
   3. FLD_STRING
   4. HDR
   5. HDR_STRING
   6. X

A description of these tatles can be found in the dataset specificaticns of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into NDBEDAC a flag is set to indicate if the ADD or CHANGE command was entered. The routine DBEDGF is called to obtain a valid fieldname. In ADD mode it must be a valid non-existent and non-reserved fieldname. In CHANGE mode it must be an existent non reserved fieldname with the exception of the fields FREEFORM and COMMENTS and it must not be a superfield nor a subfile control field. If in ADD mode a FLD structure is allocated, initialized and posted with the fieldname.

The user is prompted for the field type and the input is validated. If it is invalid, the user is given a diagnostic and prompted for a new value. If in update mode, the user is not allowed to change the field type if it affects the field length and the field appears in the fixed part of the record. The user is also not allowed to make the key field a bit field.

If there is more data in the parameter list 'TYPE' the user is prompted for an alignment value. If it is invalid, the user is given a diagnostic and prompted for a new value. If no value is entered a built in default value is assumed and posted in FLD.

The field form is prompted for and validated. If invalid, the user is qiven a diagnostic and prompted for a new value. The anchor file key field and bit field can only be of fixed form. In Update mode, the user can not change fixed field to a varying or elemental field. the necessary values are posted in FLD.

The user is prompted for a field length and the value is validated against prestored maximum values for single and multielement fields. If it is invalid, the user is given a diagnostic and prompted for a new value. The correct values are posted in FLD.

If the field is non-elemental, go prompt the user for a conversion routine, otherwise the user is prompted for an element length value if necessary. For several field types the only element length value is posted in FLD. If the element value is invalid, the user is given a diagnostic and prompted for a new value.

The user is prompted for the number of elements and the input validated. If the value is invalid, the user is given a diagnostic and prompted for a new value. A correct value is posted in FLD. The parameter unique element is prompted for and processed in the same manner.

The routine DBEDGR is called to obtain and process the conversion, formatting, validation routine names and validation argument.

At this point of adding the key field or in update mode, the rest of the parameters are ignored. in update mode the changed information is posted to the descriptor dataset by calling DBEDFL, and then go save the command string. The use is prompted if the field is to be indexed. If the answer is no then go prompt the user for associated file information, otherwise the user is prompted as to which index file the field is to appear. If no defining fieldname is entered, a new index file is created for this field. Otherwise the field is placed on the same index as that of the defining field. In the CHANGE command, if the field was already indexed on a different file, it must be deleted from that index file before it is placed on the new index file.

The user is prompted as to whether the index key is to be in either internal or external form. If no value is entered, internal is assumed. If the value is external then the user is prompted for and must enter the maximum length of the external value.

The user is prompted if the index is to be spanned. If no value is entered, it is assumed not to be spanned. At this point, the index is ready to be setup. If it is a new index a header descriptor is allocated and setup for the index, else the new information is posted to the existing header.

The routine DBEDGA is called to determine if the field is to be placed on an associate file.

The user is prompted if the field is to be on a subfile. If not go prompt the user for a subfile value as obtained, the subfile header is updated accordingly.

The user is prompted for the defining base field name if the field is to be a subfield. If no value is entered, the field will not be a subfield. If it is to be a subfield, the user is prompted for an offset value. If none is entered, a value of 0 is assumed. In case the defining base field is either RECLEN or the anchor key field the user is prompted for the particular file on which this subfield is to be placed. The user can specify the actual file name if known or indicate the type of file on which the subfield is to be placed. If ASSOCIATE or SUBFILE is specified the user is prompted for a field defining which associate file or which subfile. the subfield information is posted in FLD.

At this point all of the parameters have been entered, processed and the information posted. It is now determined which file list the field is to be placed and if not in the proper place already, threaded onto the end of that file list.

If adding the anchor key field then the fields FILEKEY, FREEFORM, and COMMENTS are setup on the appropriate files and an index file is setup for FREEFORM.

The command string is saved in the current strategy and control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

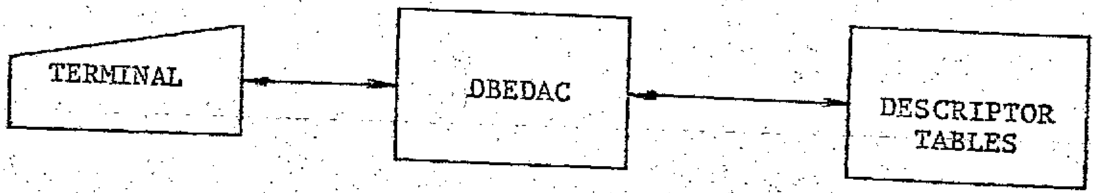


Figure 1. I/O Block diagram

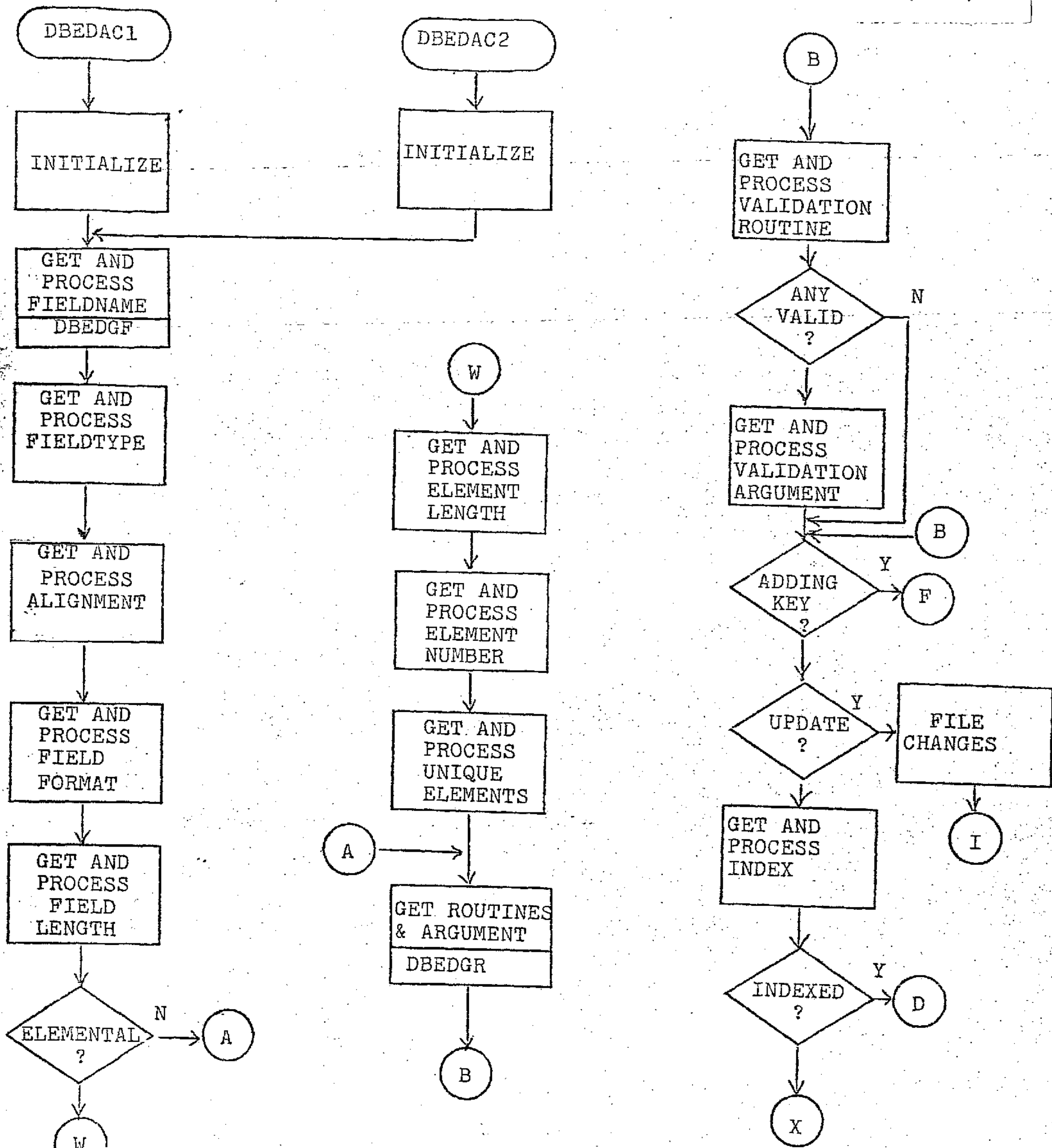


Figure 2a. Top level flowchart

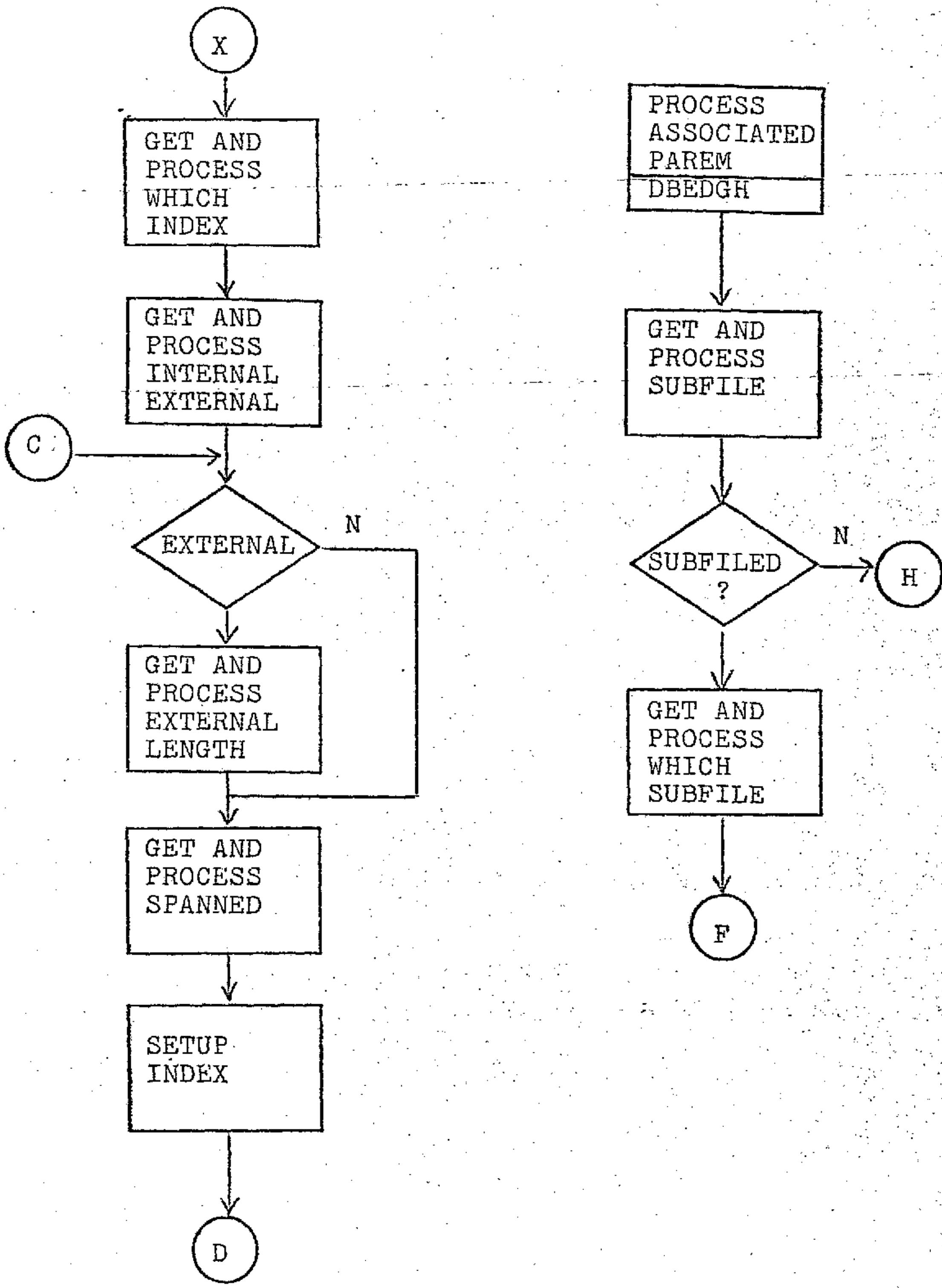


Figure 2b. Top level flowchart

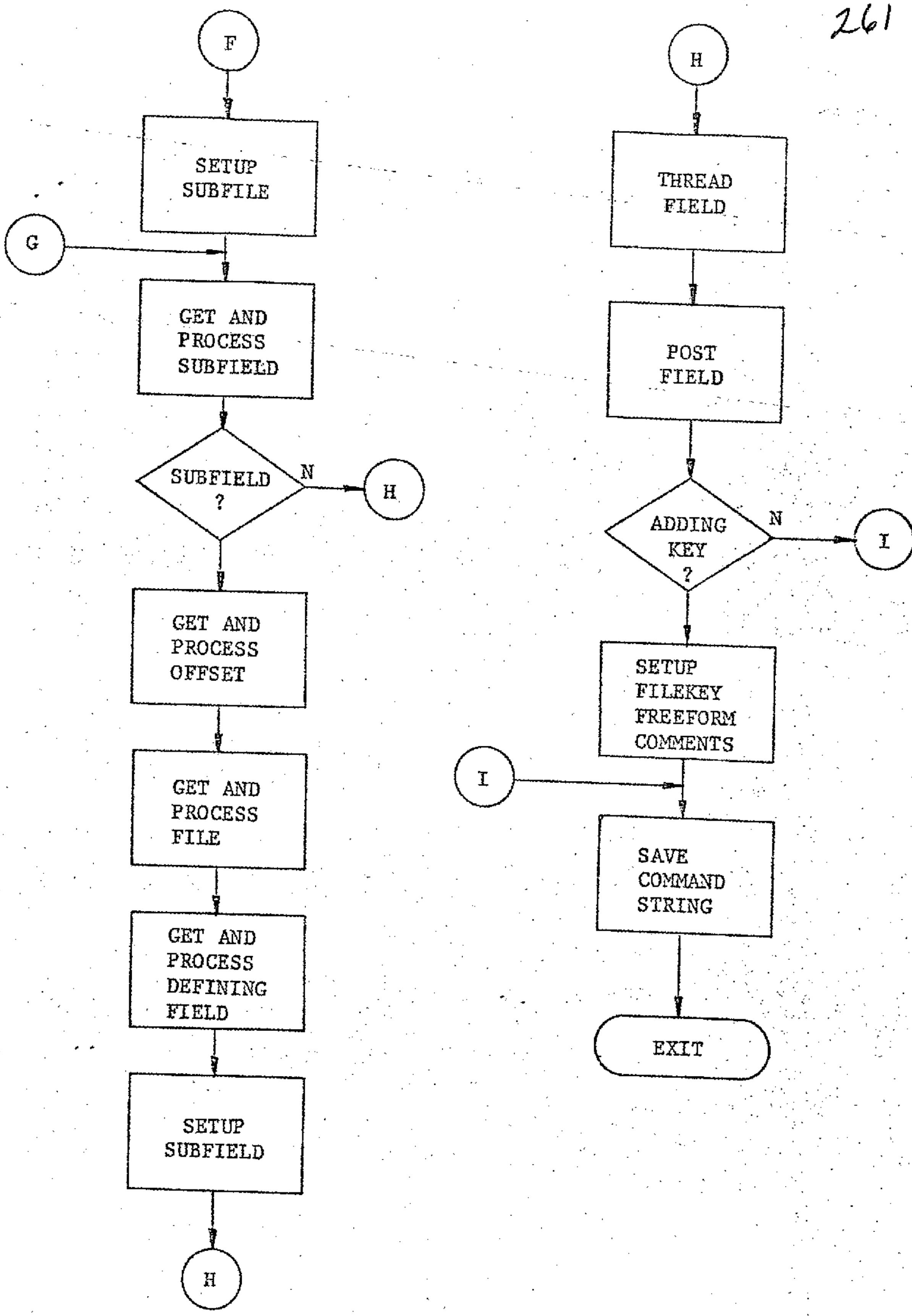


Figure 2c. Top level flowchart

TOPIC D.10 - DESCRIPTOR EDITOR - ADDLIKE - RENAME COMMANDS

A. MODULE NAME

Program-ID - NDBEDAR
Module-ID - DBEDAR

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The ADDLIKE command creates a new field descriptor exactly like an existing descriptor with a different name. The RENAME command changers the name of an existing descriptor.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Print-outs

      Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDAR:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. SECURITY
7. SUPER
8. SUPER_STR
8. SUPER_STR
9. VALID
10. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The entry points are ADDLIKE command - DBEDAR1 and RENAME command - DBEDAR2. Upon entry into either command a flag is set indicating which command was called. After which the two commands share common code for parameter processing.

Routine DBEDGF is called to obtain a valid new field name. To be valid the new field name must be of alphanumeric of at most eight characters long, must not already exist and must not be a reserved field name.

Routine DBEDGF is called to obtain a valid existing fieldname. This field must exist and must not appear in the reserved fieldname list.

If in the RENAME command, then the name of the specified existing field is changed to the specified new field name and the field name change is posted in the FIELD structure. At this point the command string is stored in the current strategy and then control is returned to the calling routine.

If in the ADDLIKE command the existing field is duplicated, the new fieldname posted in the copy.

The new fieldname is posted in the FIELD structure. The ADDLIKE command string is saved in the current strategy, after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

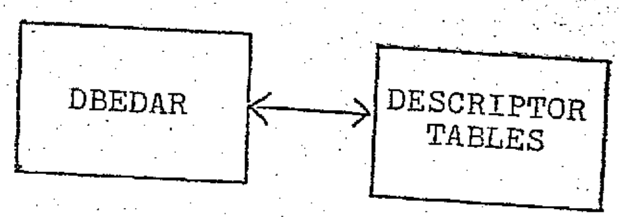


Figure 1. I/O Block Diagram

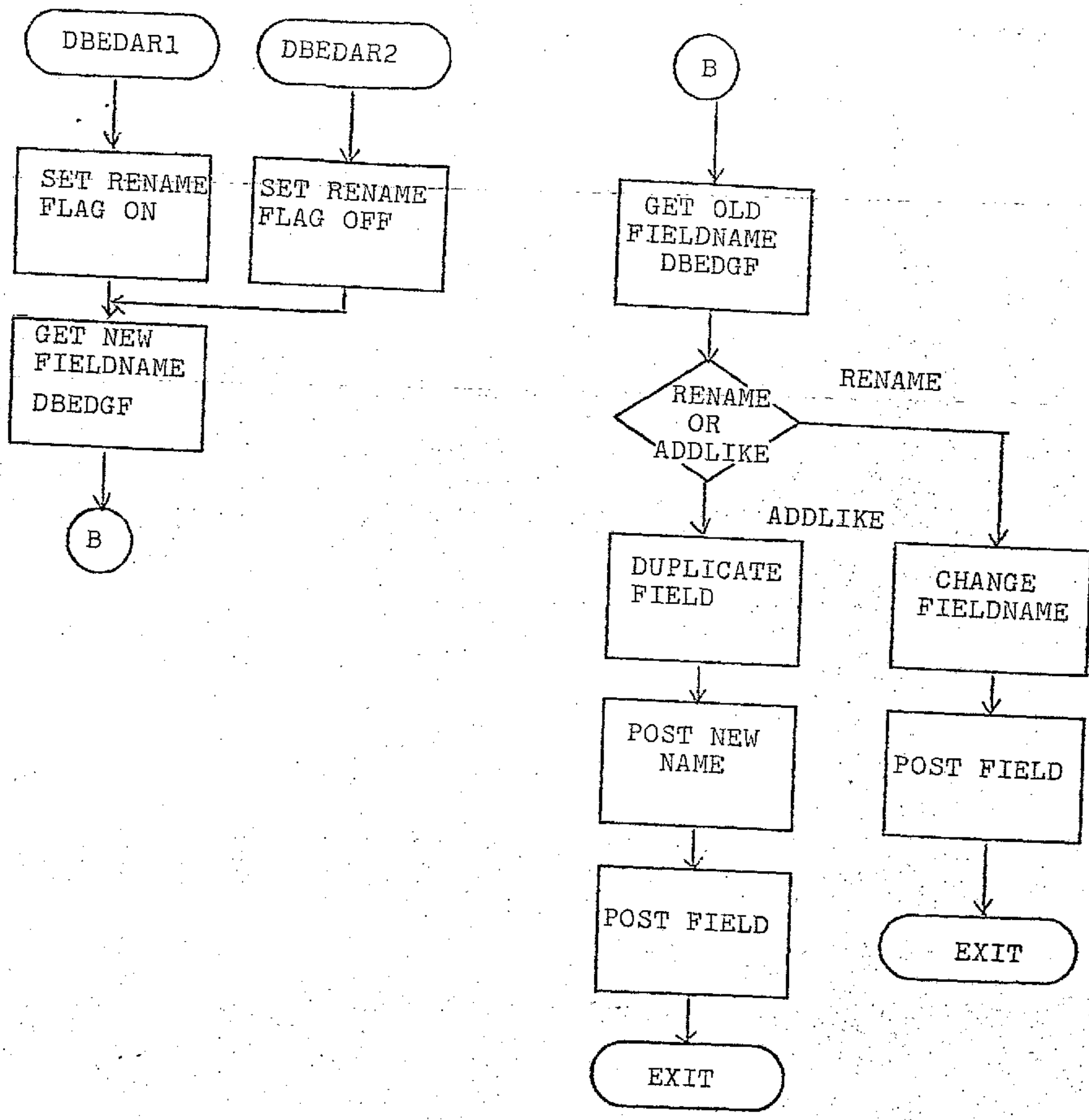


Figure 2. Top Level Flowchart

TOPIC D.11 - DESCRIPTOR EDITOR - CHKPOINT COMMAND

A. MODULE NAME

Program-ID - NDBEDCP
Module-ID - DBEDCP

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTIONS

This command is used to save the descriptor tables as they exist in memory in a SAM data set as that they may be retrieved at a future time by use of the RESTORE command and then continue to create the descriptors from that point.

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

The output file is a SAM data set named DESCRP.CHKPOINT. Refer to the data set specifications for a description of this data set.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDCP:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC_STR
7. SECURITY_STR
8. SUPER_STR
9. VALID
10. X

The description of these tables is specifications in the dataset of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into CHKPOINT, any previously existing checkpoint dataset is erased by use of ASMERSE routine. The data set record length is dynamically determined by calculating the length of that part of the X structure that must be saved, the current length of the FIELD structure and using the larger of the two values.

The data set CHKPOINT.DATASET is created and initialized by use of the routine ASMDCB to create the DCB for the data set, routine ASMFNDS to initialize the JFCB, and routine ASMOPEN to open the checkpoint data set.

The variable part of the X structure is put into the data set by use of the ASMPUT routine, and likewise the FIELD structure.

Each of the fields are saved through use of ASMPUT routine by creating the following character string: the FLD_STRING concatenated to SUPER_STR if it is a superfield, concatenated to SECURITY_STR if there is field security on this field, concatenated to VALID.ARGUMENT if the field

has a validation argument.

Each of the headers are saved through use of ASMPUT routine by creating the following character string: the HDR_STRING concatenated to RECSEC_STR if the file has record security.

The checkpoint dataset is closed by use of the routine ASMCLOS, after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

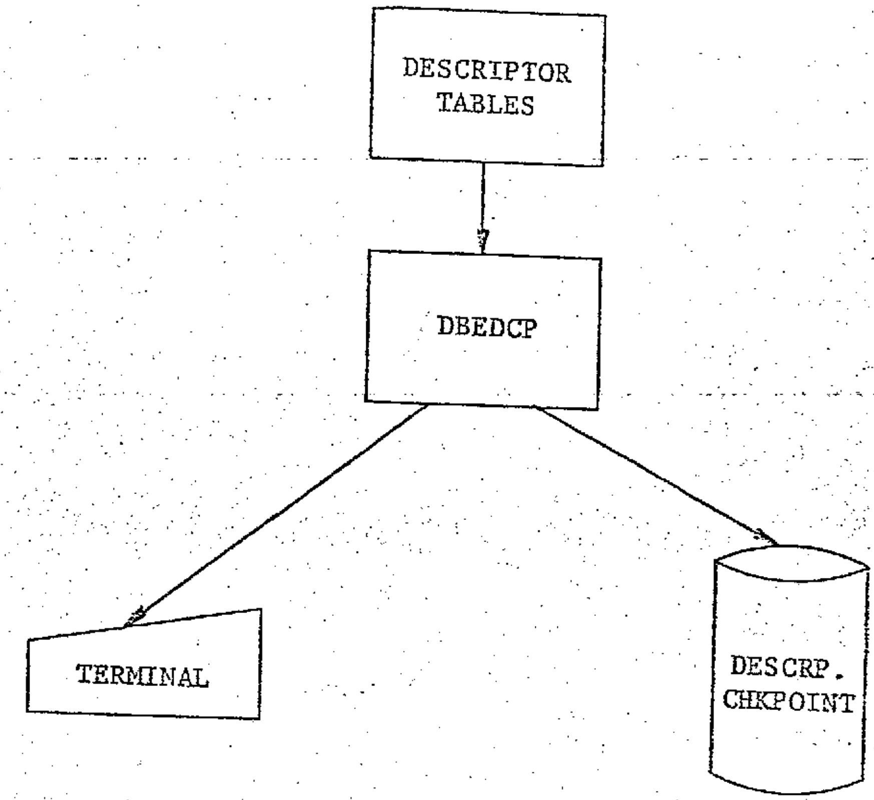


Figure 1. I/O Block diagram

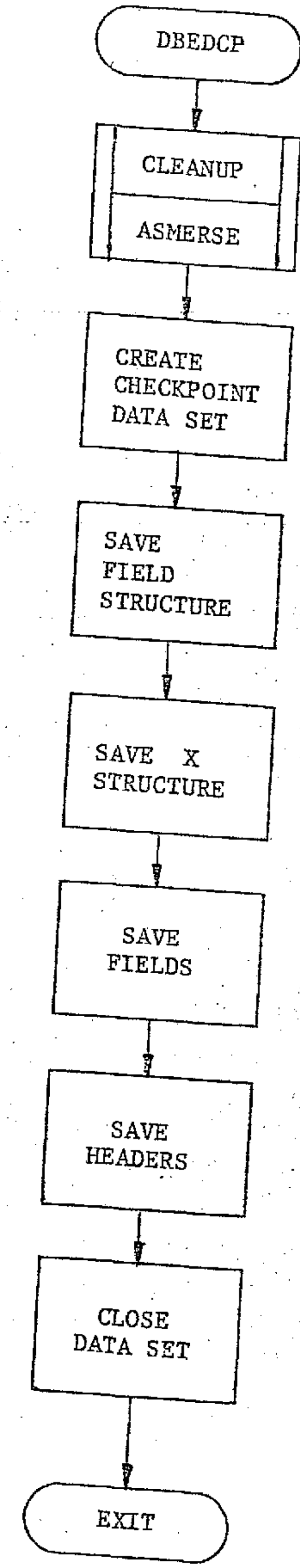


Figure 2. Top level flowchart

TOPIC D.12 - DESCRIPTOR EDITOR - CREAT SUB COMMAND

A. MODULE NAME

Program-ID - NDBEDCS
Module-ID - DBEDCS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This routine is used to define and setup the necessary field to create a subfile.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Print-Outs

      Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDCS:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into CREATSUB, module DBEDGF is called to obtain a valid subfile control fieldname. To be valid this field name must not be longer than six characters long and be a valid alphanumeric character string. The following field names are then created. The subfile kev field name, the subfile parent key field name, and the subfile record security field name. Of the above mentioned four fieldnames, none must exist and none must be a reserved field name for the entered subfile control field name to be valid.

The user is prompted for the maximum number of subfile records per anchor file record that may be loaded into the subfile. The number must be less than or equal to 1325. If the number is valid, processing continues, else the user is given a diagnostic and prompted for a new number. This number then becomes the number of elements on the subfile control field.

Routine DBEDGA is called to determine if the subfile control field is to appear on an associate file.

The subfile control field, the subfile key field, and the subfile parent key field are now created and posted with the proper values. The subfile control field is placed in the varying field list of either the anchor file or the appropriate associated file. The subfile key field and the parent field are placed in the fixed list of the

appropriate subfile.

The afore mentioned field names are placed in the reserved field name list. The command string is saved in the current strategy, after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

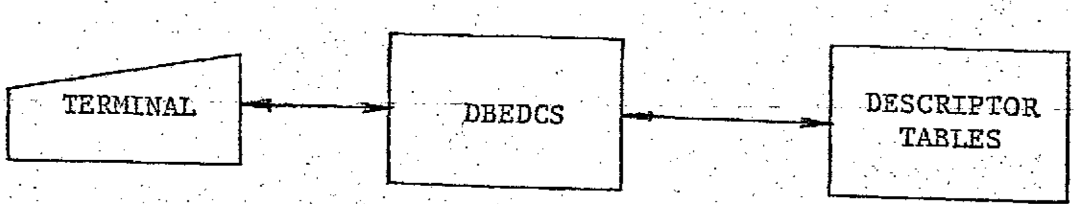
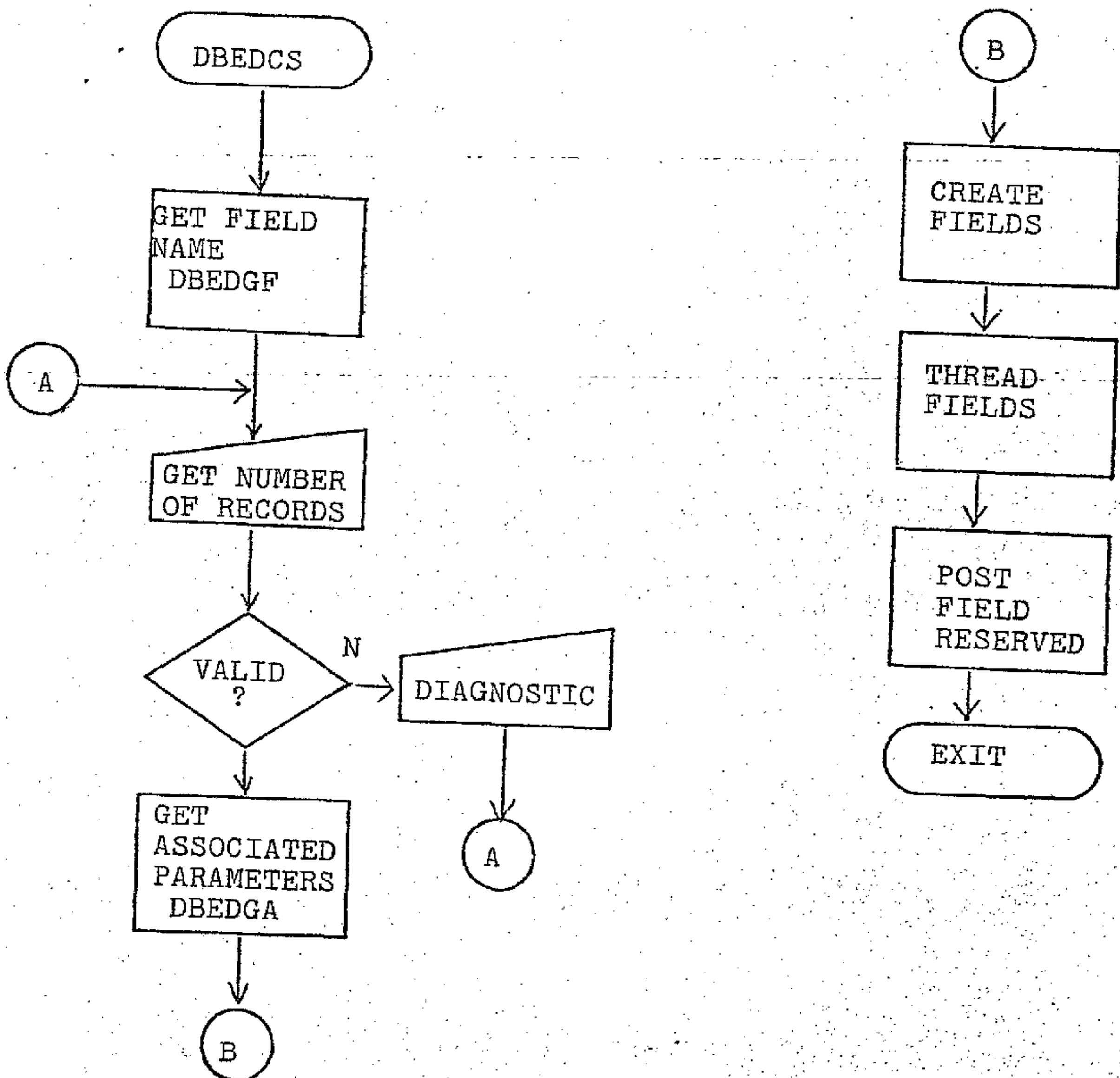


Figure 1. I/O Block diagram


DBEDCS
GET FIELD NAME DBEDGF
A
GET NUMBER OF RECORDS
VALID ?
N
DIAGNOSTIC
A
GET ASSOCIATED PARAMETERS DBEDGA
B
B
CREATE FIELDS
THREAD FIELDS
POST FIELD RESERVED
EXIT

TOPIC D.13 - DESCRIPTOR EDITOR - END COMMANDS

A. MODULE NAME

Program-ID - NDBEDDE
Module-ID - DBEDDE

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is the entry point into the Descriptor Editor. It prompts for and processes Descriptor Editor commands and calls the appropriate command routine. The END command is used to terminate Descriptor Editor processing and return control to the maintainence director.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDDE:

1. FIELD
2. X
3. VERBTAB

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Module DBEDIN is called to set up mode of operation and all of the tables necessary to the running of the descriptor editor.

The user is prompted for his next Descriptor Editor command. If the command is invalid as determined by a search of the verb table, the user is given a diagnostic and prompted for a command string.

If the command is not END, then the appropriate command is called by use of the CALL routine when control is returned, the user is prompted for his next command.

If the command is END then if the user has not filed his corrections, additions, or changes, he is prompted informing him such and asked if he really wants to terminate the Descriptor Editor. If the anser is no then the user is prompted for his next Descriptor Editor command, else the Descriptor Editor run is terminated.

At termination each field storage and each header storage area is released. The FIELD and X structures are then released. Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I and TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

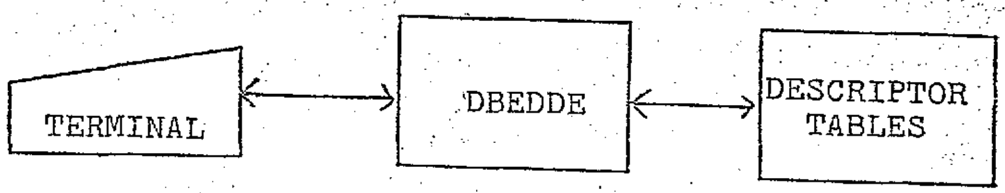


Figure 1. I/O Block diagram

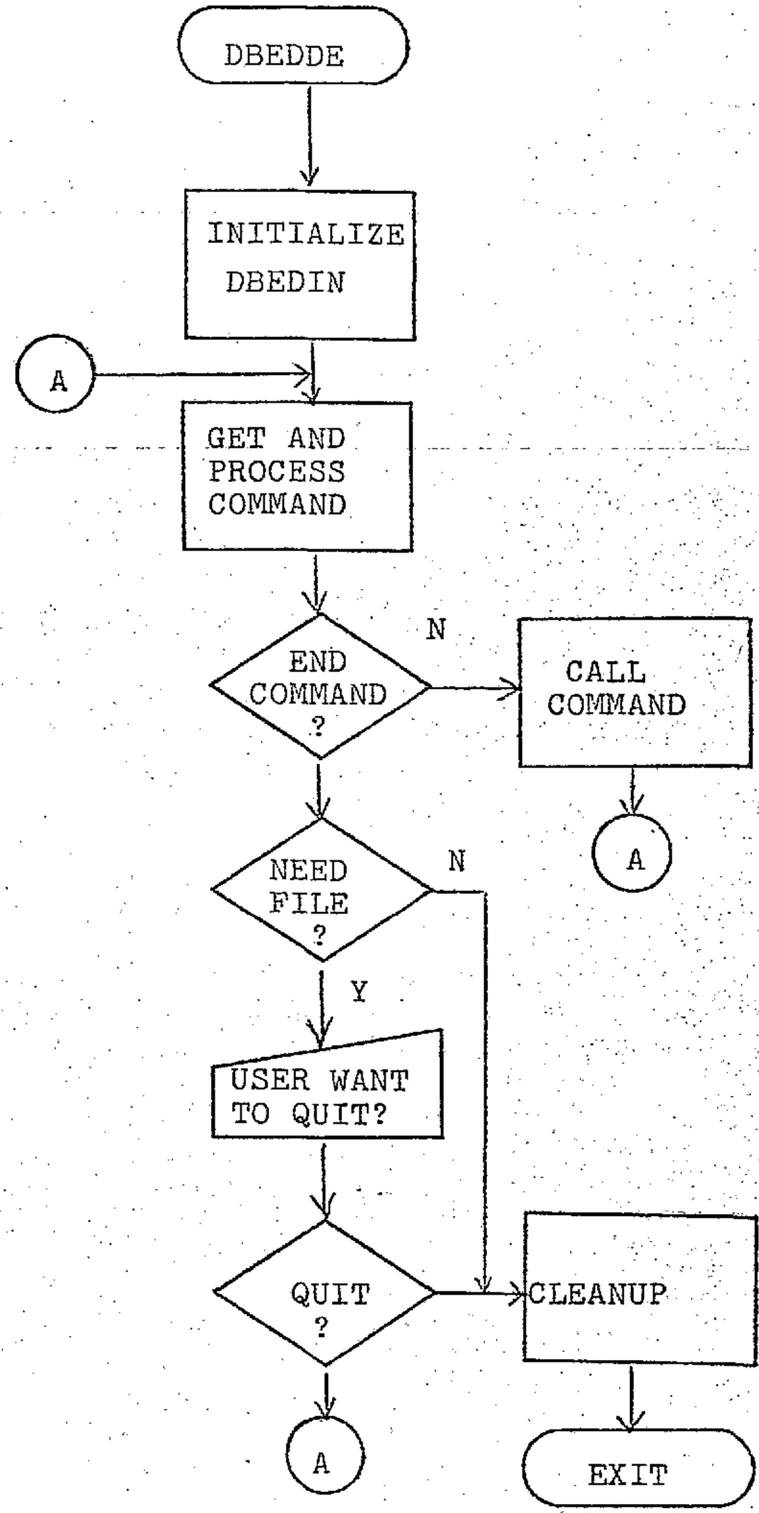


Figure 2. Top level flowchart

TOPIC D.14 - DESCRIPTOR EDITOR - DISPLAY INTERNAL COMMAND

A. MODULE NAME

Program-ID - NDBEDDI
Module-ID - DBEDDI

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is a debugging tool used to display the various external descriptor tables (DESCTAB), by their internal name, field descriptors by their field name and header descriptors by their file ids.

DISPLAYI DISTYPE=<I,F,H>,DISNAME=structure-name

where:

DISTYPE is the type of variable to be displayed I for internal, F for field descriptor and H for file or header descriptor.

DISNAME is the name of the variable to be displayed. For internal mode the following structures may be displayed.

ERRORFILE
FIELD
FLD
HDR
RECSEC
SECURITY
SUPER
VALID
FLD_COMMENTS
FLD_FREEFORM
FLD_RS
FLD_SUBCNTRL
FLD_SUBID
FLD_SUBPK
HDR_ASSOC
HDR_INDEX
INIT_FLD
INIT_HDR
INIT_RECSEC
INIT_SECURITY
INIT_SUPER

IO_FLD
IO_HDR
IO_RECSEC
IO_SECURITY
PLEX
RESERVEC
XS
X

For field mode the name of the field to be displayed is supplied. For header mode the file suffix id is entered.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

1. FIELD
2. FLD
3. HDR

4. RECSEC
5. SECURITY
6. SUPER
7. VALID
8. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into NDBEDDI the user is prompted for the display type. If the display type value is not 'I', 'F', or 'H' the user is given a diagnostic and prompted for a new value.

Depending on the display type the user is prompted for either an internal structure name, a field name, or a header id. If the internal structure name is not contained in the list of names in the module function section, or the field does not exist or the file does not exist, the user is given a diagnostic and prompted for a new display name value.

When displaying an internal name, a label variable is used, one label for each structure that may be displayed. At each of these pieces of code, a generalized display subroutine is called to display the desired type of structure passing the address of the particular structure to be displayed. This is done for all structures except for the structures PLEX, ERRORFILE, and XS. A word about these display procedures later. The information from the structures PLEX and ERRORFILE is setup and displayed. The display of the X structure is a service of calls to the different display procedures, one for each minor structure of X to be displayed.

When displaying a field descriptor, a call to the procedure DIS_FLD is called to display the proper FLD structure. If the field is a superfield, has a validation argument, or has field security, calls are made to the routines DIS_SUPER, DIS_VALID, and DIS_SECURITY to display the proper structures. This is done to display all of the

information associated with the field.

When display a header descriptor, a call is made to DIS_HDR to display the proper HDR structure and if the file has record security, a call is made to DIS_RECSEC to display the appropriate record security information.

After the information has been displayed, control is returned to the calling routine.

For displaying the actual desired data several internal procedures are set up, one for each type of structure. They are DIS_FIELD, DIS_FLD, DIS_HDR, DIS_RECSEC, DIS_RESERVED, DIS_SECURITY, DIS_SUPER, DIS_VALID, and DIS_XS. These procedures build the output information in a work area in predefined formats. The information is output to the terminal thru use of the TS PROMPT facility. The output consists of a title line followed by the data usually displayed beneath the title line.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I and DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

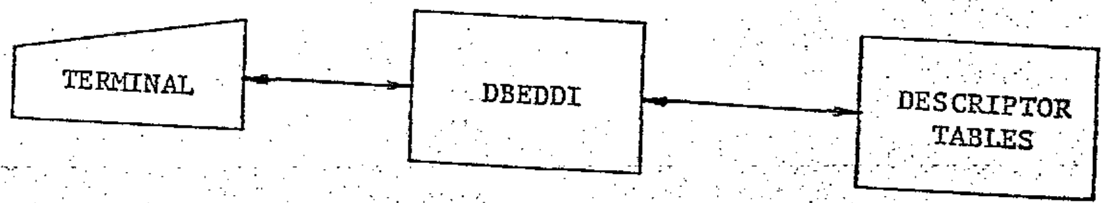


Figure 1. I/O Block diagram

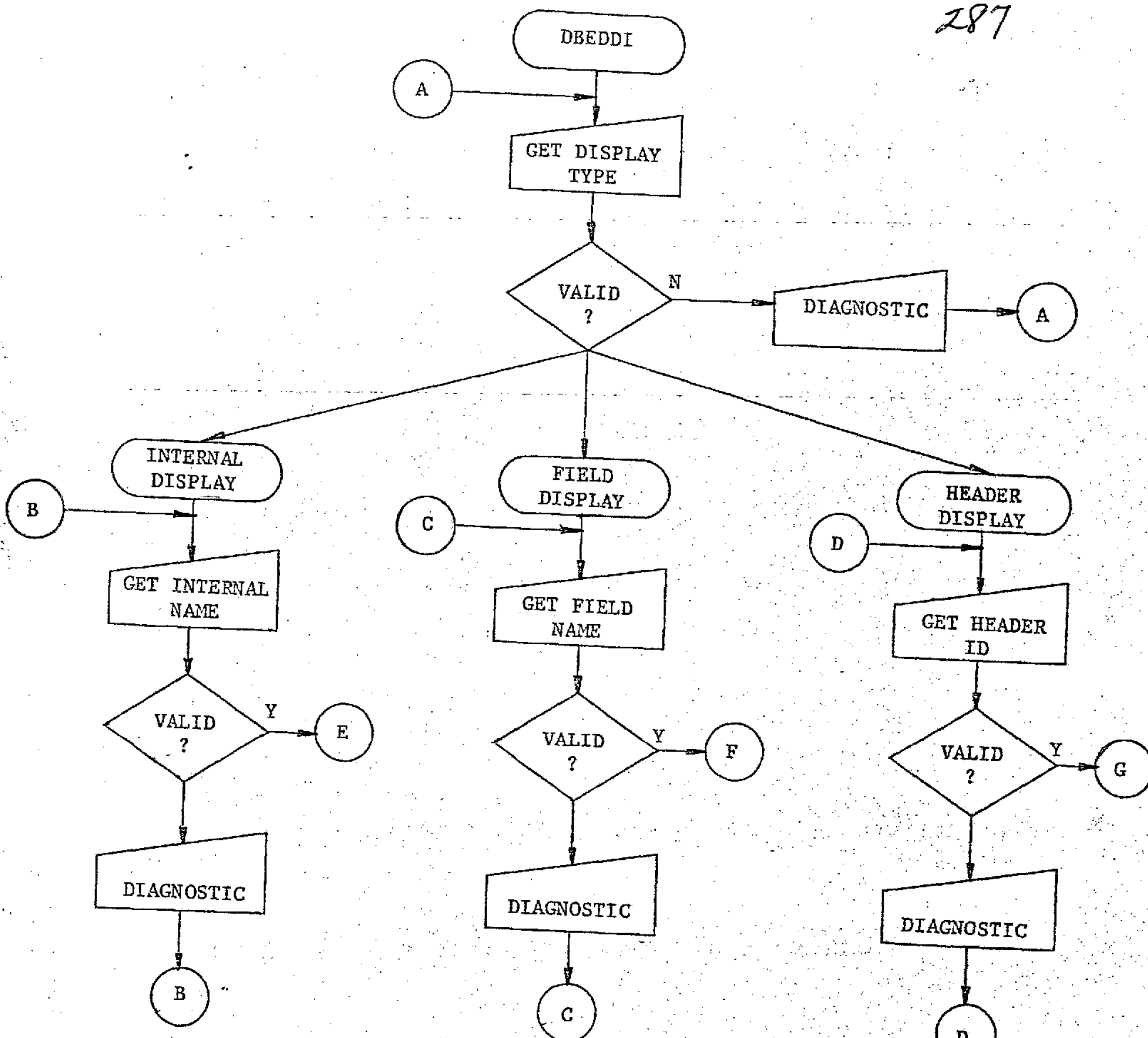


Figure 2a. Top level flowchart

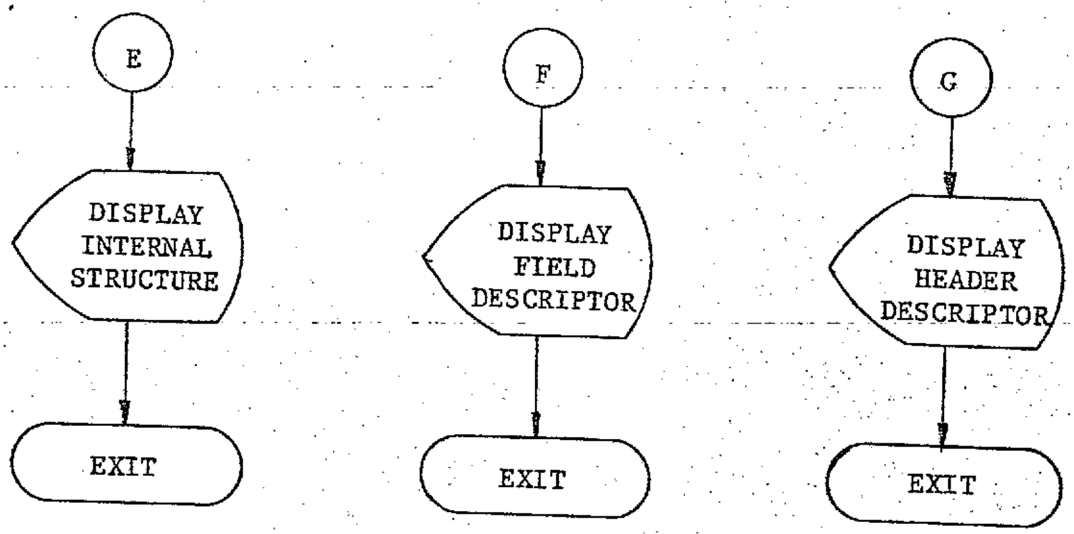


Figure 2b. Top level flowchart

TOPIC D.15 - DESCRIPTOR EDITOR - DELETE FIELD COMMAND

A. MODULE NAME

Program-ID - NDBEDDL
Module-ID - DBEDDL

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is used to delete a previously defined field descriptor.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDDL.

1. FIELD
2. FLD
3. HDR
4. SUPER
5. X

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Routine DBEDGF is called to obtain a valid fieldname. To be valid, the field must exist. If the field name appears in the reserved list, then it must be a subfile control field and there must be no other fields on this subfile to be valid.

A further check is made to determine if the field to be deleted is a component of any superfields or is the defining base field for any subfields. If so, the user is told of all superfields and all subfields that make use of this field. The user is then prompted for a new field name value. If here then the field can be deleted.

At this point, the internal delete entry point is defined. If the field name to be deleted does not exist, control is returned to the calling routine. Otherwise a pointer is set to the field to be deleted. At this point delete forms common code.

If the field appears on an associate or subfile or is indexed, then the appropriate file descriptor counts are updated. If the associated file or index file is depleted of fields, the file headers are deleted, and the file ids made available for reassignment.

At this point the field is deleted by the internal delete field routine. If the deleted field is a subfile control field, the subfile key field and the parent key field are also deleted.

The next field in the list to be deleted is now processed in the afore mentioned manner. After all of the fields have been processed, the command string is saved in the current strategy, if it was the delete command that was called. Control is then returned to the calling routine.

The internal procedure DELETE FIELD is used to release the work areas containing the field

information, and to post the deleted field list if this field exists on the disc storage version of the descriptor file.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

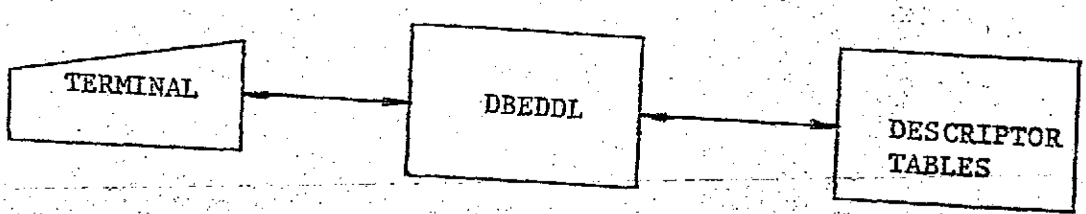


Figure 1. I/O Block diagram

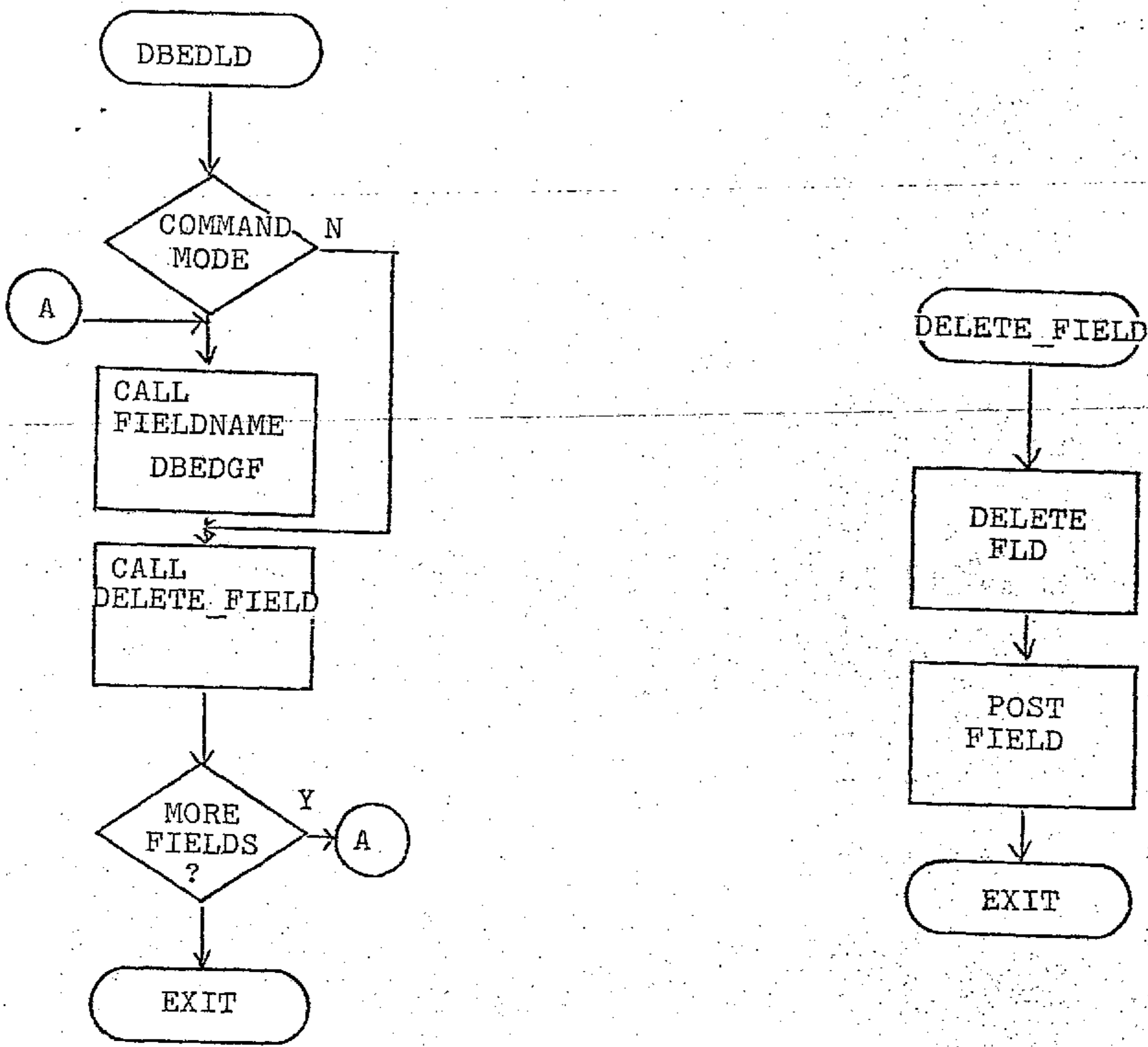


Figure 2. Top level flowchart

TOPIC D.16 - DESCRIPTOR EDITOR - DISPLAY FIELD COMMAND

A. MODULE NAME

Program-ID - NDBEDDP
Module-ID - DBEDDP

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This routine is used to display the information defining a field.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input File

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The various pieces of information are displayed on the screen one item per line preceded by a descriptive title. Refer to the dataset specifications for a description of this display format.

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDDP:

1. FIELD
2. FLD
3. HDR
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

At the command entry point the paging information structure is allocated and initialized and routine DBEDGF is called to obtain the fieldname to be displayed.

At the paging entry point, the paging information is set to point to the proper page to be displayed and then join common code with the command entry point.

At the start of the common code the number of the next item to be displayed is retrieved from the paging information and a branch is taken to the appropriate code to obtain the next piece of field information. If there is no information for this item number, the next item is pointed to and processed as above. After the line of information is built, it is placed in the screen buffer.

If there is more room in the buffer, the next item is pointed to and processed as above. Once the screen is full and there is more information to be output in the forward direction, a paging entry point is setup and next page information is posted in the paging information structure. The buffer is then flashed to the screen after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

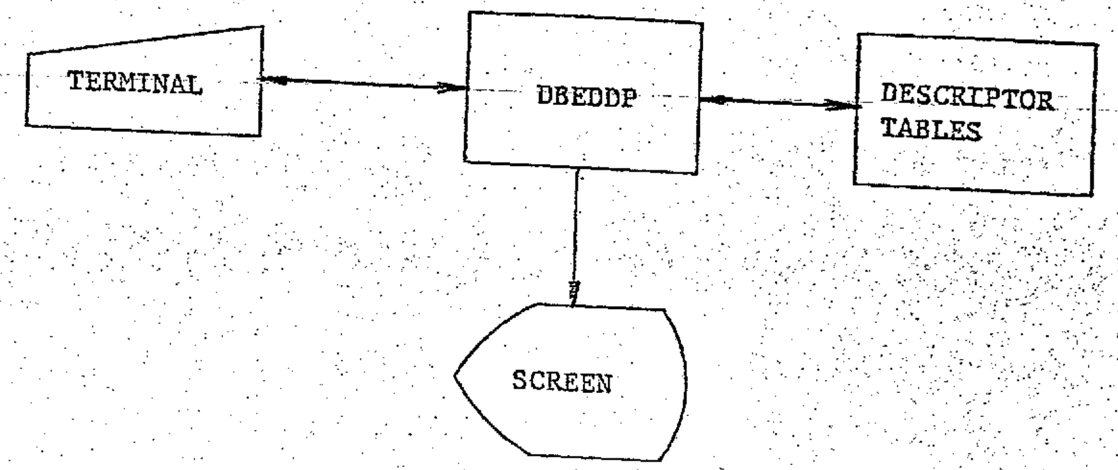


Figure 1. I/O Block Diagram

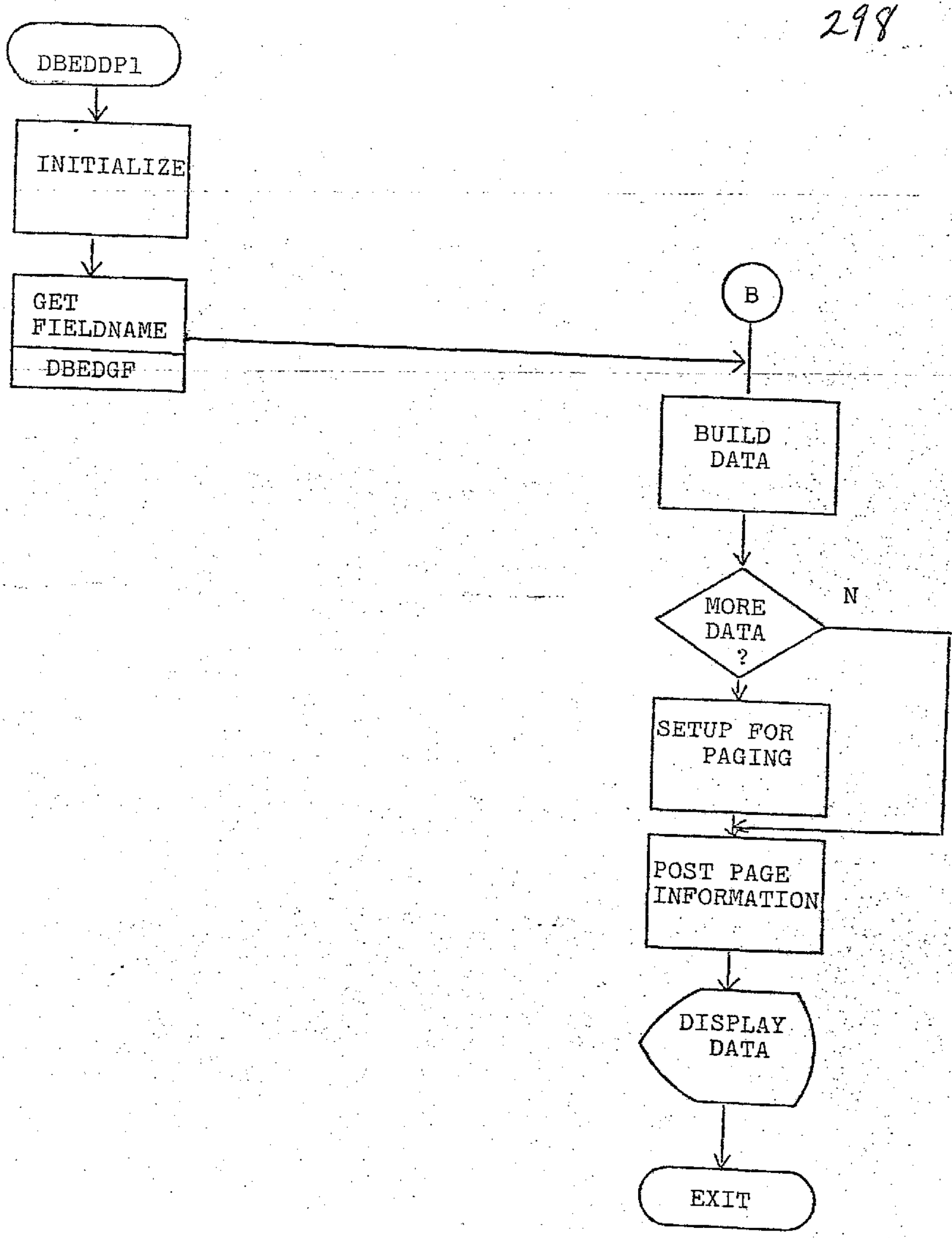


Figure 2a. Top Level Flowchart

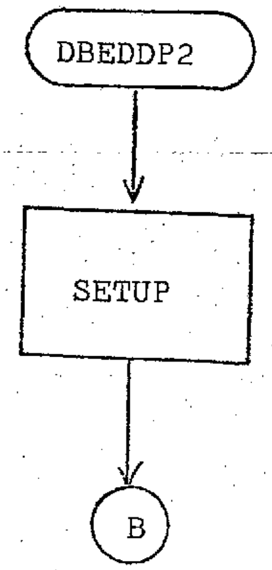


Figure 2b. Top Level Flowchart

TOPIC D.17 - DESCRIPTOR EDITOR - Initialization

A. MODULE NAME

Program-ID - NDBEDIN
Module-ID - DBEDIN

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module performs all of the initialization necessary for the running of the Descriptor Editor. It is called by the Descriptor Editor director.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

There are no input files for the Descriptor Editor when in the CREATE mode and the user is creating a new set of descriptors. However, when in UPDATE mode, or when the user is continuing the creation of a previously entered set of descriptors, the previously created descriptor file is used as an input file. The description of this file is found in the dataset specifications of the DWB.

3. Output Data Sets

a. Output Files

Not Applicable

b. On Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDIN:

1. FIELD
2. FLD
3. HDF
4. RECSEC
5. SECURITY
6. SUPER
7. VALID
8. X

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The descriptor file indicated is opened for input to determine if the file exists. If the file exists and in CREATE mode, routine DBEDID1 is called to load the descriptors. If in UPDATE mode and the file does not exist, the user is given a diagnostic and prompted for a new file name.

The verb table is allocated and initialized to the proper verb table copy. The routine DBUSER is called to setup any additional user defined commands.

If in CREATE mode and no file exists, the user is

prompted for the anchor key field. The routine DBEDAC1 is called to process and setup the anchor key field.

The user is prompted for the descriptor mode. If the response is valid, flags are set indicating the mode, and a pointer is set to the appropriate verb table copy. If the mode is invalid, the user is given a diagnostic and prompted for a new mode value.

The X structure is allocated and initialized. The initialization consists of setting the various pointers in X to NULL. he FIELD structure is allocated and its pointers set to NULL.

If RESTORE mode is indicated, DBEDRT is called to restore the checkpointed descriptor. If no restore errors occurred, the go setup the verb table. If there were restore errors, or CREATE or UPDATE mode were indicated, the file name is retrieved from the MFCB.

Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I and TSPL/I statements

2. Suggestions and Techniques

Not Applicable

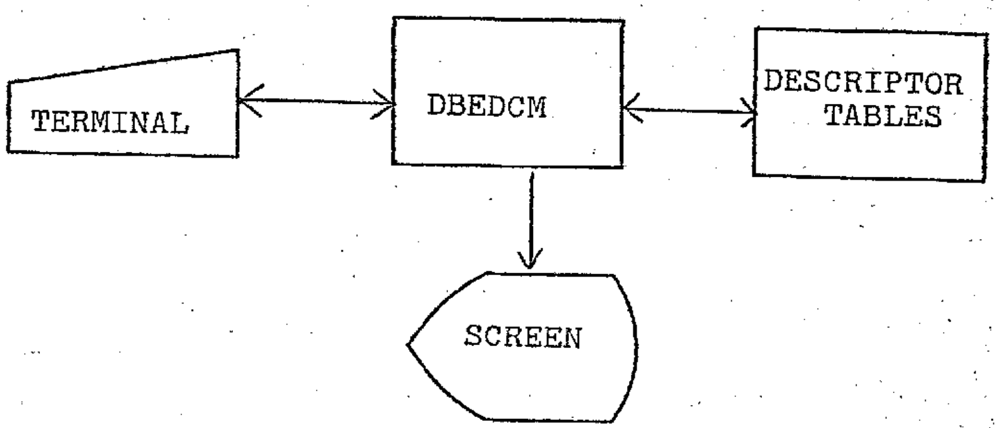
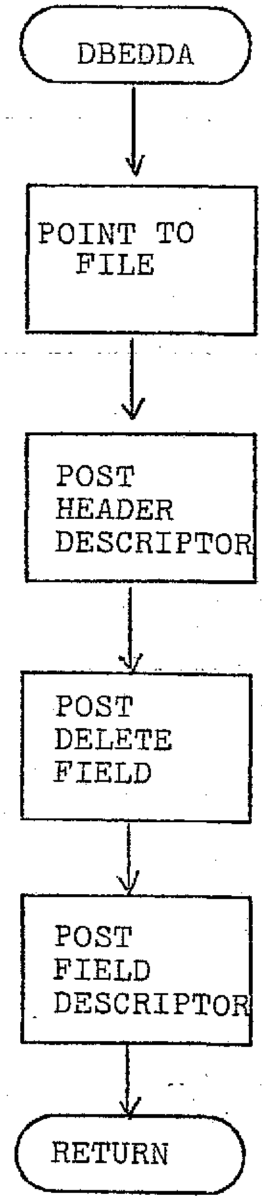
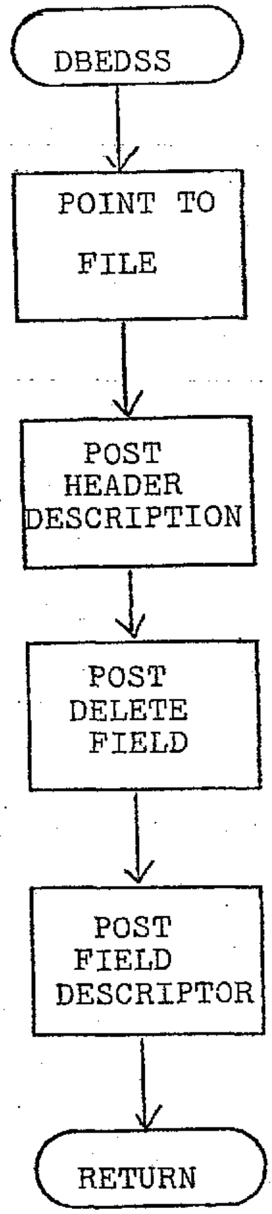


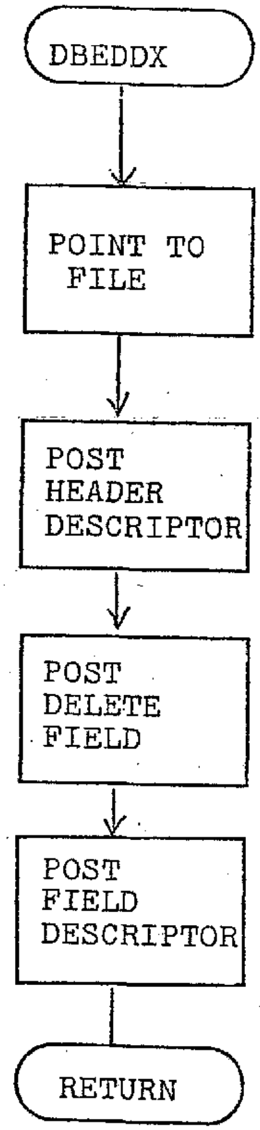
Figure 1 - I/O Block Diagram

CM4


DBEDDA
POINT TO FILE
POST HEADER DESCRIPTOR
POST DELETE FIELD
POST FIELD DESCRIPTOR
RETURN


DBEDSS
POINT TO FILE
POST HEADER DESCRIPTION
POST DELETE FIELD
POST FIELD DESCRIPTOR
RETURN

CM6


DBEDDX
POINT TO FILE
POST HEADER DESCRIPTOR
POST DELETE FIELD
POST FIELD DESCRIPTOR
RETURN

307
CM7

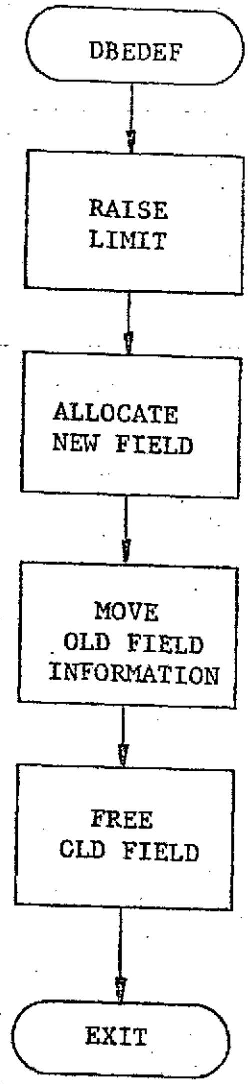
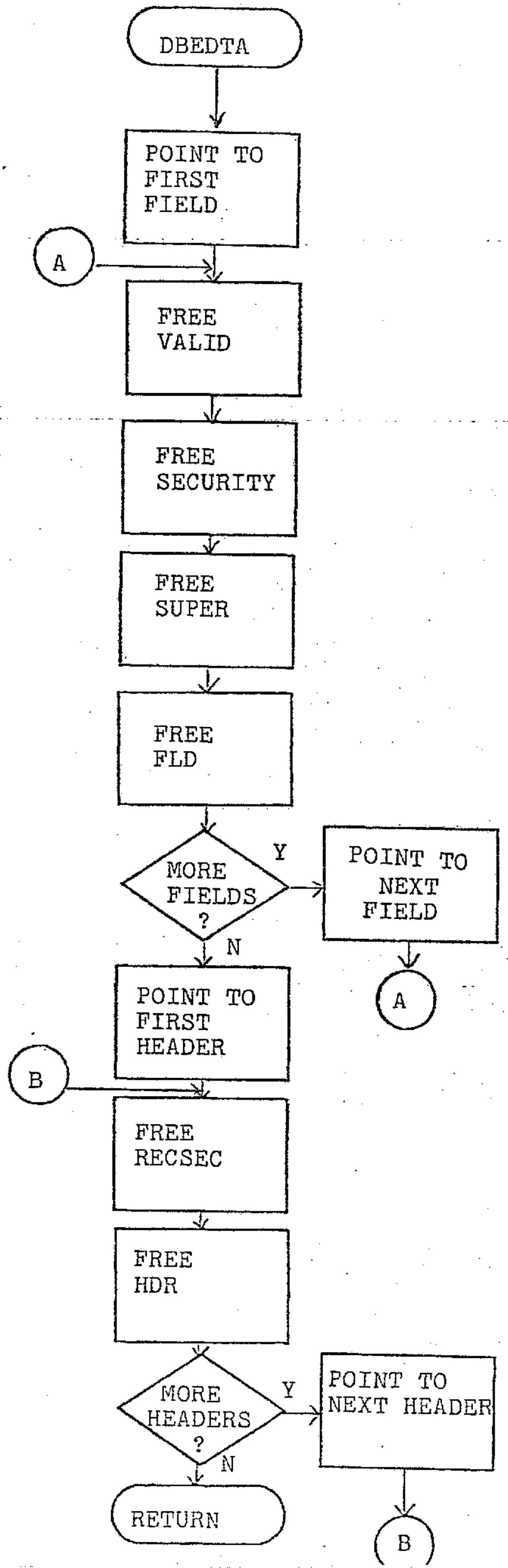
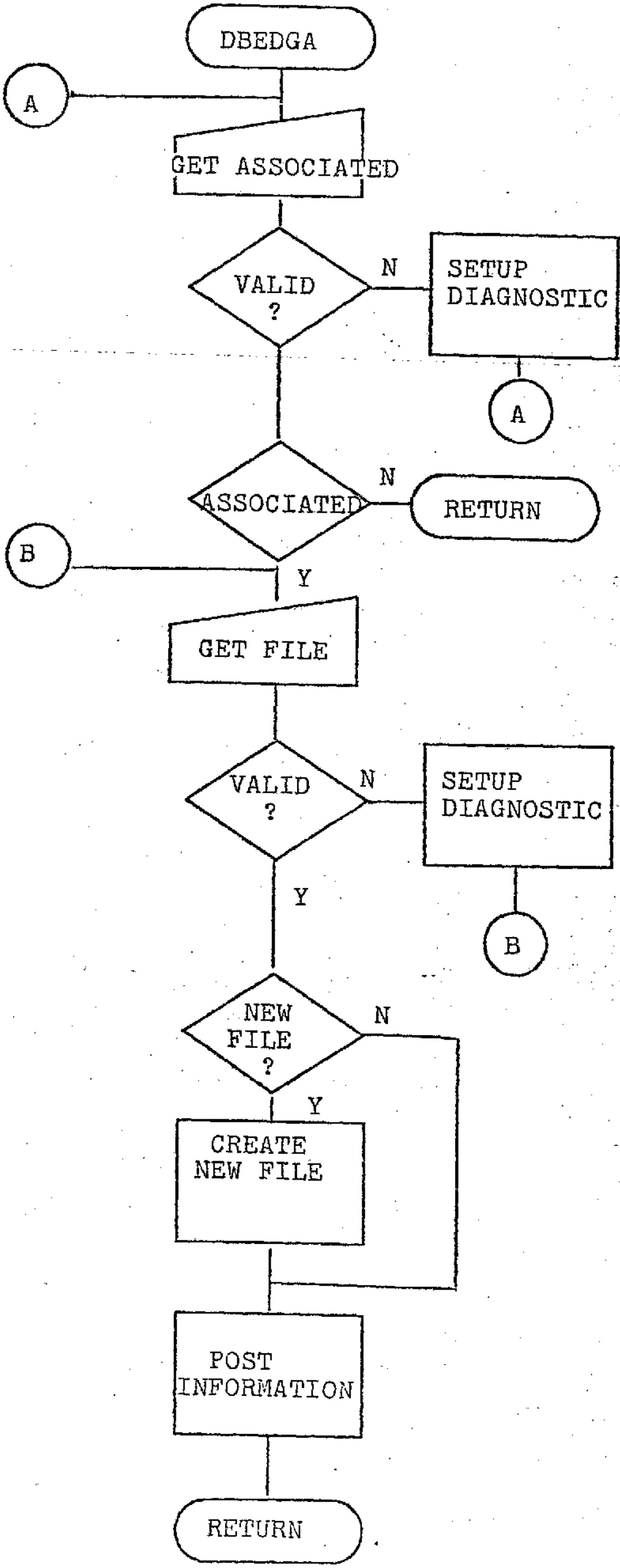
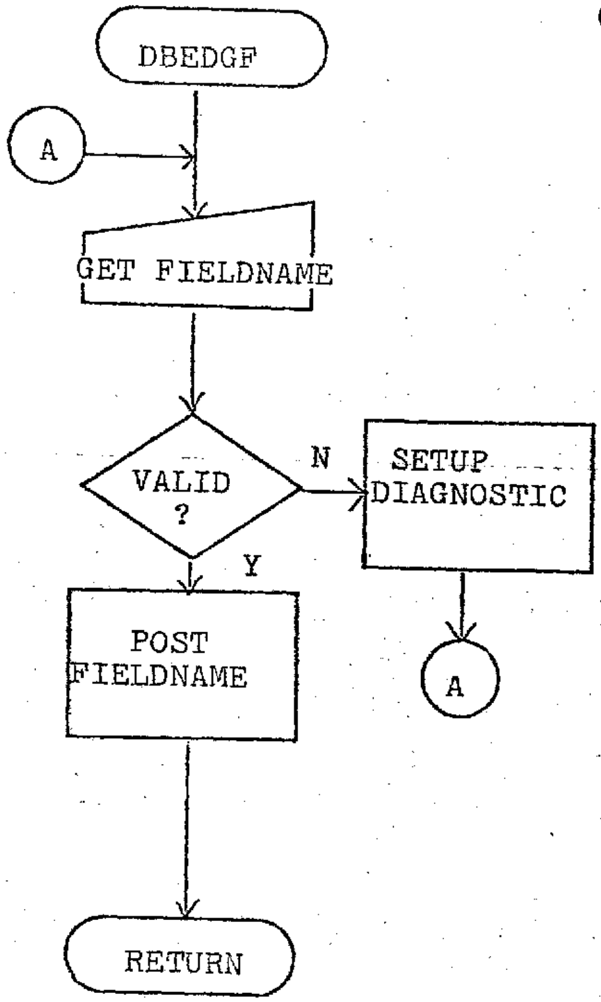


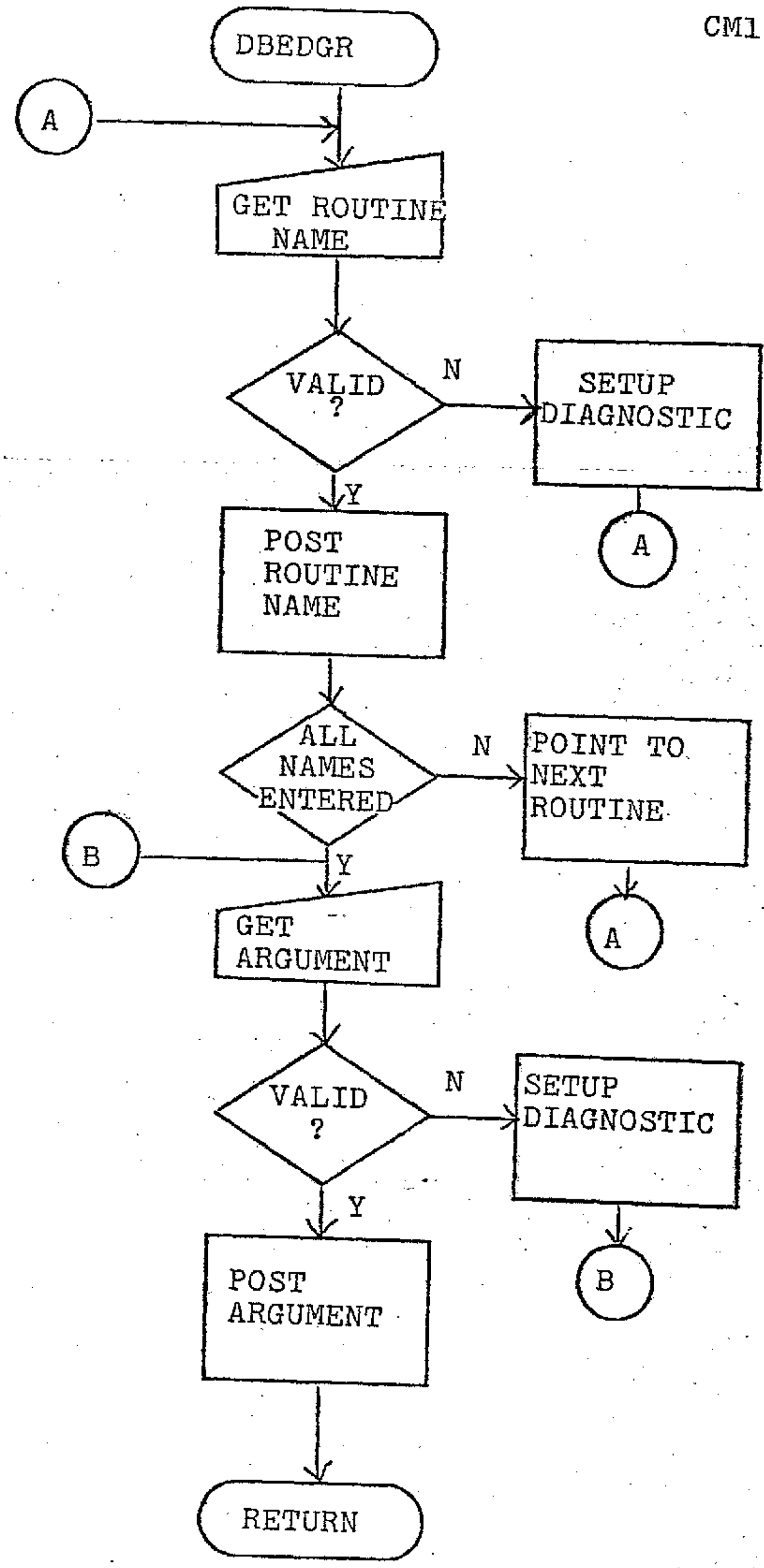
Figure 2. Top Level Flowchart


DBEDTA
POINT TO FIRST FIELD
A
FREE VALID
FREE SECURITY
FREE SUPER
FREE FLD
MORE FIELDS ?
Y
N
POINT TO NEXT FIELD
A
POINT TO FIRST HEADER
B
FREE RECSEC
FREE HDR
MORE HEADERS ?
Y
N
POINT TO NEXT HEADER
RETURN
B

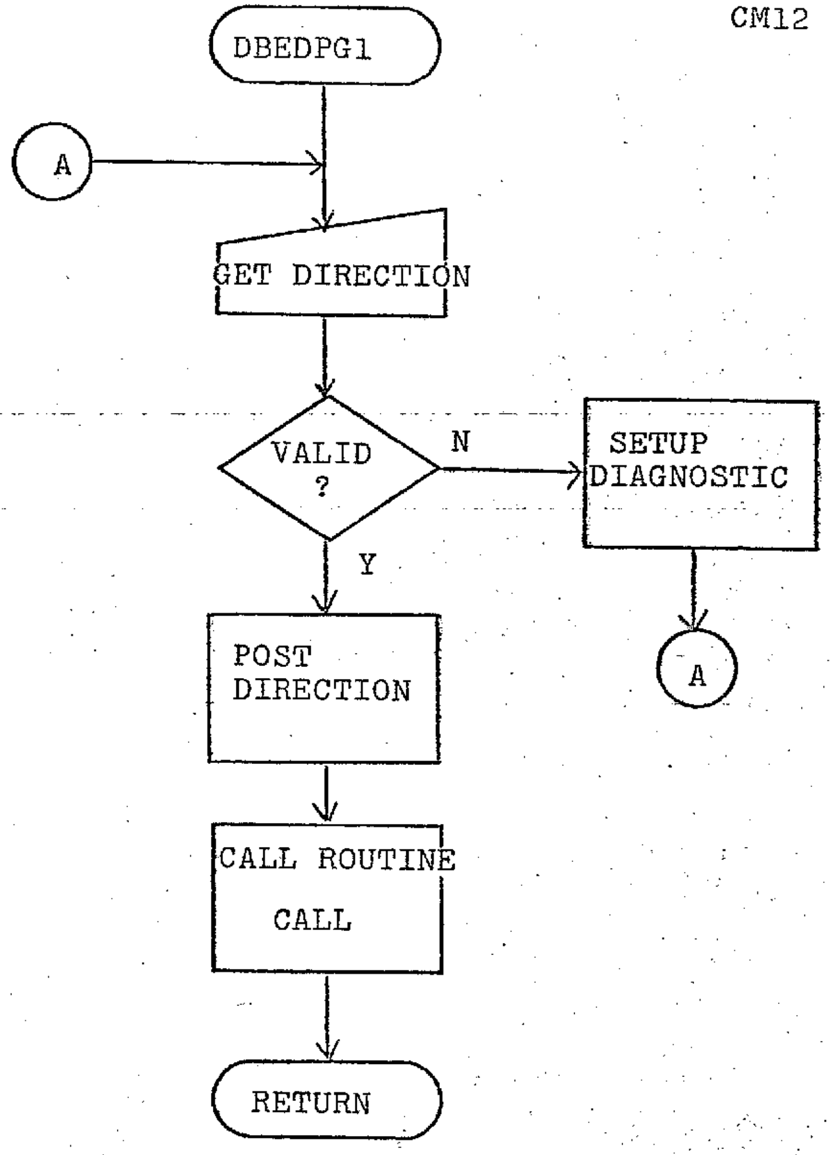
CM9

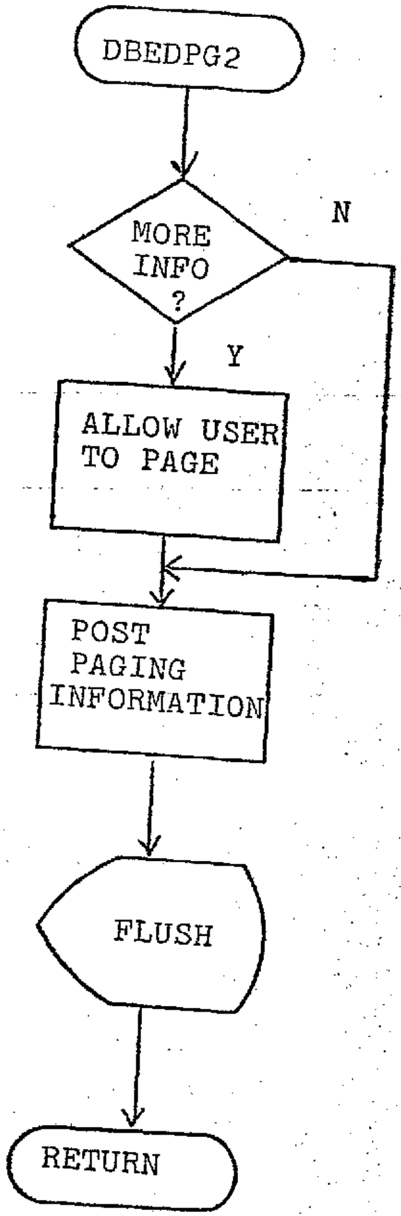

DBEDGA
A
GET ASSOCIATED
VALID ?
N
SETUP DIAGNOSTIC
A
ASSOCIATED
N
RETURN
B
Y
GET FILE
VALID ?
N
SETUP DIAGNOSTIC
Y
B
NEW FILE ?
N
Y
CREATE NEW FILE
POST INFORMATION
RETURN


DBEDGF
A
GET FIELDNAME
VALID
?
N
SETUP
DIAGNOSTIC
Y
POST
FIELDNAME
A
RETURN


DBEDGR
A
GET ROUTINE NAME
VALID ?
N
SETUP DIAGNOSTIC
A
Y
POST ROUTINE NAME
ALL NAMES ENTERED
N
POINT TO NEXT ROUTINE
A
B
Y
GET ARGUMENT
VALID ?
N
SETUP DIAGNOSTIC
B
Y
POST ARGUMENT
RETURN

CM12


DBEDPG1
A
GET DIRECTION
VALID ?
N
SETUP DIAGNOSTIC
A
Y
POST DIRECTION
CALL ROUTINE
CALL
RETURN


DBEDPG2
MORE INFO ?
N
Y
ALLOW USER TO PAGE
POST PAGING INFORMATION
FLUSH
RETURN

TOPIC D.18 - DESCRIPTOR EDITOR - FIELDS COMMAND

A. MODULE NAME

Program-ID - NDBEDFD
Module-ID - DBEDFD

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

In CREATE mode the FIELDS command outputs the names of the fields thus far created. In UPDATE mode the descriptor descriptor names are displayed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      The fieldnames are placed on the screen, the number of names per line determined by dividing the screen width by 20.

   c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDFD:

1. FIELD
2. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

If in CREATE mode, a pointer is set to the FIELD structure, otherwise in UPDATE mode the pointer is set to an internal list containing the descriptor descriptor field names.

At the paging entry the proper page number is set up in the paging information structure.

At this point, the code becomes common for both the command and paging entry points. The number of the next field name to be displayed is obtained from the paging information structure. Two control loops are set up, one to build every line to fill the screen and the other to fill each line of the screen.

If there is more information to be displayed, the paging entry point is set up. The paging information structure is posted, the buffer is flushed to the screen and control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

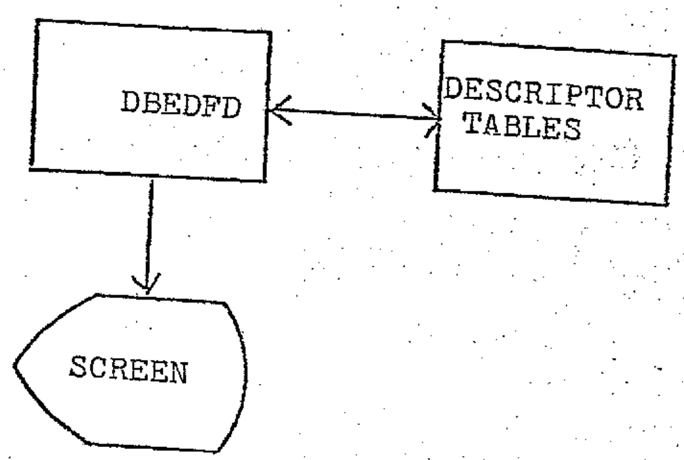


Figure 1. I/O Block Diagram

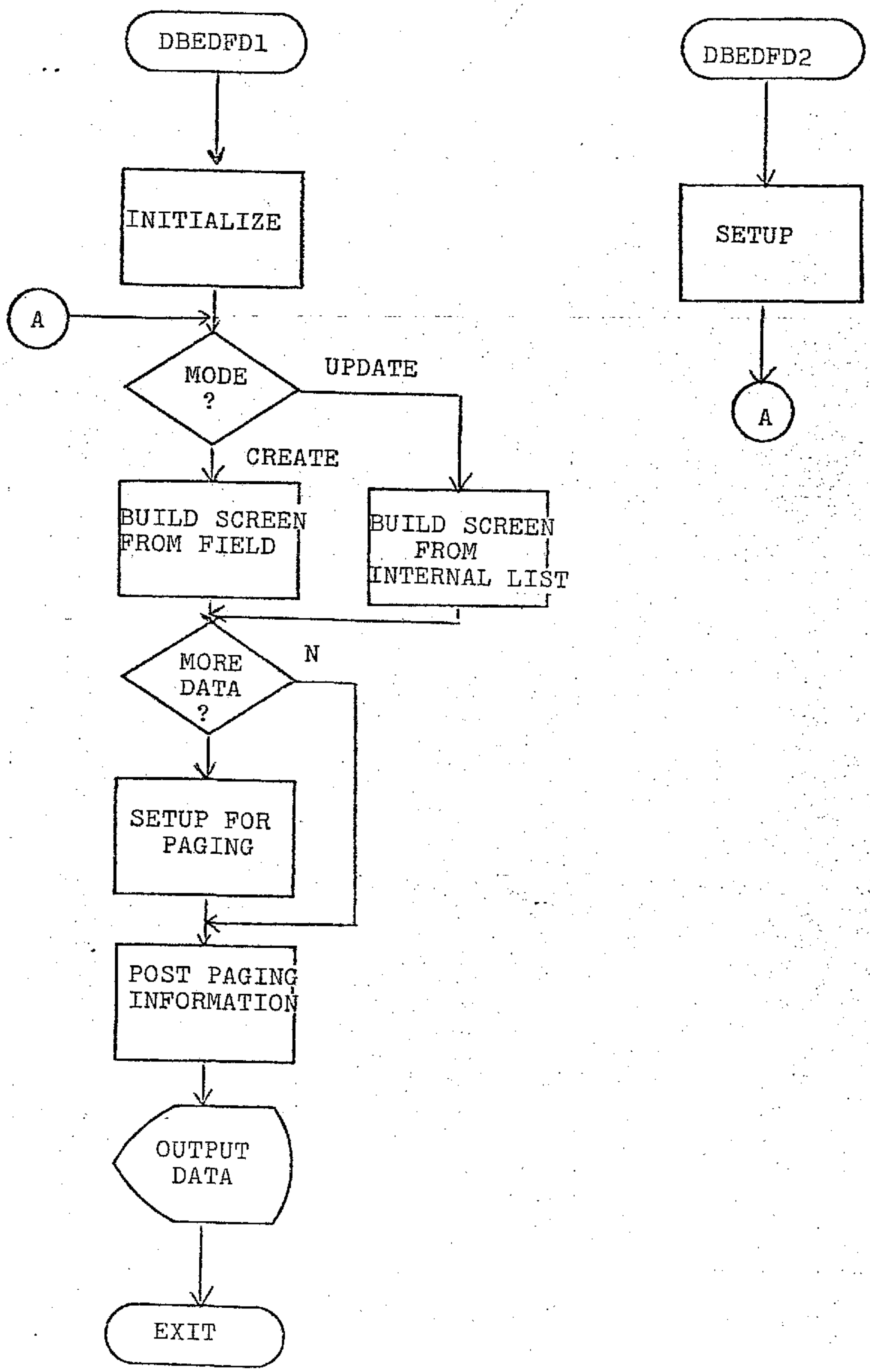


Figure 2. Top Level Flowchart

TOPIC D.19 - DESCRIPTOR EDITOR - FILE COMMAND

A. MODULE NAME

Program-ID - NDBEDFI
Module-ID - DBEDFI

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is used to place those additions and/or changes from the descriptors in core to the descriptor file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

The descriptor file is a region ISAM dataset containing all the information necessary to completely define the data base.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDFI:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC
7. SECURITY
8. SUPER
9. VALID
10. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entering FILE command, if just one descriptor record is to be updated, the appropriate file identified is setup, the file opened and the descriptor record is updated after which control is returned to the calling routine. Otherwise all the descriptor information is to be filed to the descriptor file. The user is prompted for the parameter DESCOK and the value saved for posting each header record. If the input value is in error the user is given a diagnostic and prompted for a new value.

If the anchor key field needs to be deleted, it is deleted from the anchor and all associate files. The delete file list is then processed deleting the header, RECLEN key field and when applicable the parent key field of all files listed.

For outputting descriptor information, the files are processed in the following order: anchor, all associate files, then all subfiles. If the file does not exist on disc the RECLEN field is written out. If it is a new region and the file is either the anchor file or an associate file the

anchor key field must be written out for a new subfile the subfile key field and parent key field are written out. If these or any other fields already exist on the file then only the changes if any, to these fields are written out. Record security if any is then written out.

As these fields are output the field position value for each field in maintained and updated. This value is then placed in the FLDPOSIT position for each field.

The packed bit fields for the file are then processed in the order in which they appear in the list. They are packed four to a byte and the field position and field length indicating which byte and where in the byte respectfully the field can be found. After all packed bits fields are processed, the fixed fields for the file are processed shipping over the key field, parent key field and record security field where applicable. Then all varying fields are processed in order.

If it is an anchor or associate file all descriptors if any are set up and processed, Then the header record is setup and processed and the file closed. The next file is processed in this manner until the anchor file, all associate files and all subfiles are processed.

The index files are processed next. If it is a new file the RECLEN field is written out. Each field to be indexed on this file is located, setup and written out. The anchor or key field on the appropiate subfile key field is setup and written out. If the index file already exists then only those changes applicable are written out to the dataset. Each index file is processed in this manner until all index files have been processed.

After all the fields have been processed the various external structures are marked indicating that the descriptor data is on the dataset. The command string is saved in the current strategy and control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I and TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

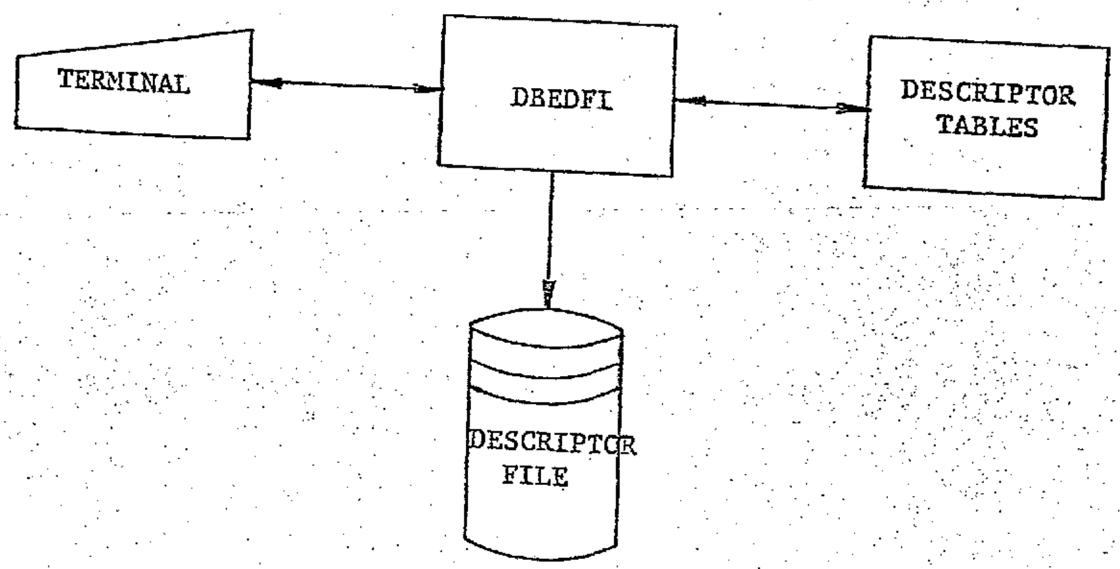


Figure 1. I/O Block Diagram

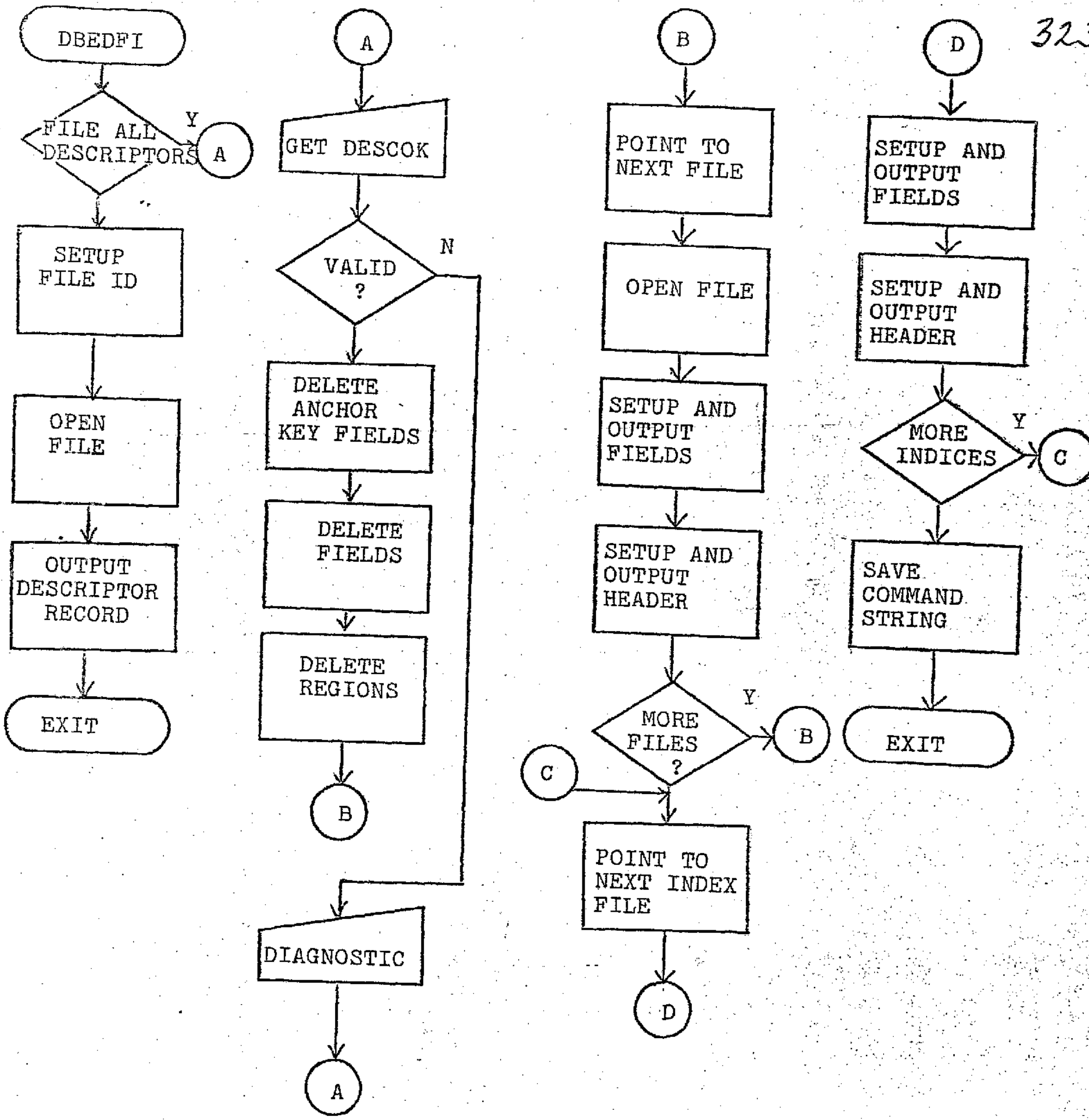


Figure 2. Top Level Flowchart

TOPIC D.20 - DESCRIPTOR EDITOR - SUPERFIELD COMMAND

A. MODULE NAME

Program-ID - NDBEDSU
Module-ID - DBEDSU

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The SUPERFLD commands allow the user to define a superfield descriptor.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Print-Outs

      Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDSU:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. SUPER
6. VALID
7. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into SUPERFLD, routine DBEDGF is called to obtain a new fieldname.

A FLD structure is allocated and initialized. Routine DBEDGR is called to obtain any conversion, formatting and validation routines and validation argument.

The user is prompted for a list of field names which are to be the superfield components. Each component is processed in the following manner. If no internal external indicator is present, external form is assumed. If an indicator is present, it is seperated from the field name. If the indicator is invalid, the user is given a diagnostic and prompted for a new component value. To be valid the component fieldname must be the name of an existing field. In addition, for a field having more than one component, the component list is limited to at most one multielement field and all if any subfile components must be from the same subfile. If the component fieldname is invalid, the user is given a diagnostic and prompted for a new component value.

After all the components are entered, and processed, they are saved in a SUPER structure and the pointer stored in the FLD structure.

Next it is determined on which descriptor file the

superfield is to be placed. If all the components are from one file, then the superfield descriptor is placed in the descriptor region. If the components are all from one associate file and one subfile and the subfile is defined off of that associate file, the superfied descriptor is placed in that associate descriptor region. All other superfield descriptors are placed in the anchor descriptor file.

The SUPERFLD command string is saved in the current strategy and control is then returned to the calling routine.

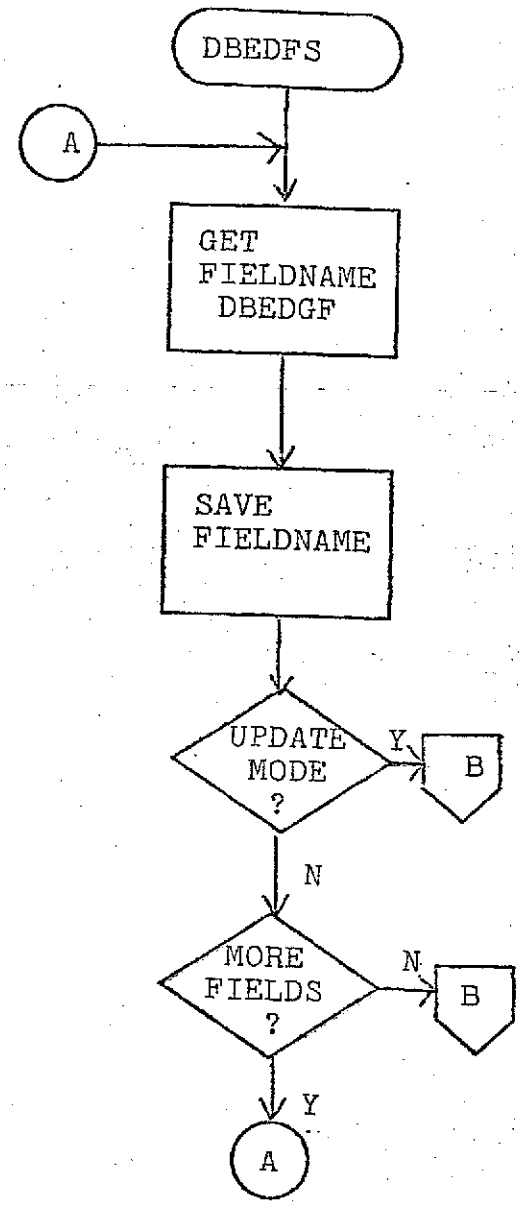
F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable


DBEDFS
A
GET FIELDNAME DBEDGF
SAVE FIELDNAME
UPDATE MODE ?
Y
B
N
MORE FIELDS ?
N
B
Y
A

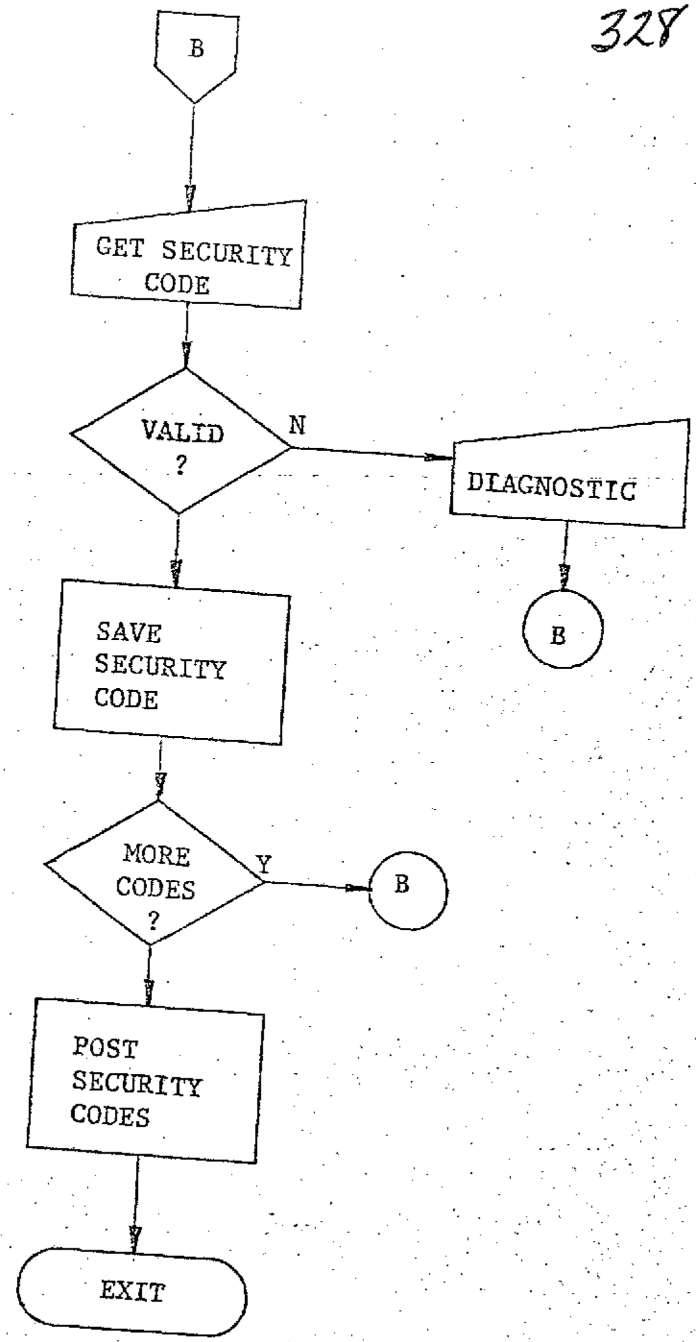


Figure 2b. Top Level Flowchart

TOPIC D.21 - DESCRIPTOR EDITOR - LOAD DESCRIPTORS MODULE

A. MODULE NAME

Program-ID - NDBEDLD
Module-ID - DBEDLD

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

In create mode the load module loads and sets up all field and header descriptor information. In update mode the load module loads the desired descriptor record, including file descriptors and dummy descriptor records.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The descriptor file is a region ISAM dataset containing all the information necessary to completely define the data base.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDLD:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC
7. SECURITY
8. SECURITY_STR
9. SUPER
10. VALID
11. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into DBEDLD if all descriptors are to be loaded, the anchor file is first pointed to, otherwise the appropriate file identifier is set up. If call from REVIEW command branch to retrieve the appropriate header on field descriptor fields as the file has been opened and the appropriate descriptor read into core.

In update mode any fields which have been loaded and still exist in work areas are released. This is a control so that no more than one field descriptor can be loaded at any one time. Note: this is not true for header descriptor.

The next descriptor region is opened starting with the anchor region and the descriptor header record read in. The header fields are obtained and all bit switches converted to an alphabetic character. A HDR structure is allocated and the header information saved therein. If the file has record security, the security codes obtained, placed in a

RECSEC structure and hooked up to the HDR structure.

If in update mode, the desired field descriptor record is read in, otherwise the next sequential field descriptor is read in. If not in review mode, it must be determined if the field is a dummy descriptor. If it is then a list of file ids is built eventually containing all of the descriptor regions on the file once all of the field descriptors on the anchor file have been processed. This list is built from non-blank entries in the ASSOCFIL, INVFILE and SUBFILE descriptor fields. If the field is a dummy, and in update mode, the correct file is pointed to and a branch goes to open the file and read the desired field descriptor. In create mode, this record is skipped and the next descriptor record in the region is read.

If this field descriptor is saved, all of the field descriptor bit field values are translated to an alphabetic character.

The field validation argument, if any, is obtained and saved. If the field is a superfield, the component values are obtained and saved. Likewise, if the field has security, the security codes are obtained and saved.

A FLD structure is allocated and the field information saved therein. The field name and pointer are posted in the next available slot in the FIELD structure, and if in create mode, the FLD structure is chained to the end of the proper file list.

When the anchor region is finished. A list of all existing descriptor regions is complete. The next descriptor region in that list is selected and loaded as described.

In review mode once the desired descriptor record from the desired descriptor region has been processed, as the correct non dummy field descriptor has been loaded in update mode, control is returned to the calling routine.

In create mode a search is made through all file lists to locate all subfields. For each subfield, the defining base field is located and the base field name and offset are posted in the subfield FLD structure.

The fields within the file lists are ordered by their field positions within each file list with all subfields and superfields appearing at the end of the crdered lists. Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

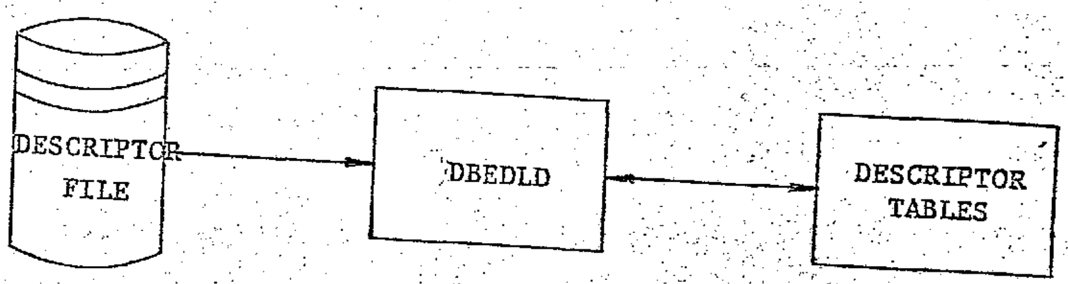


Figure 1. I/O Block Diagram

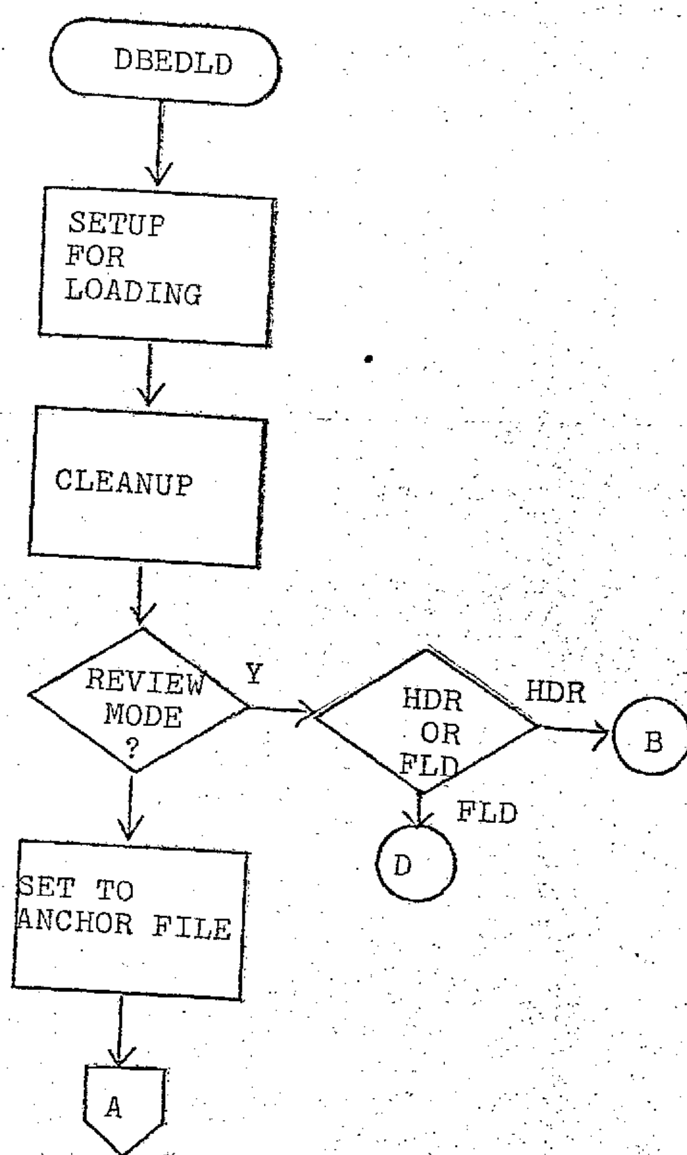


Figure 2a. Top Level Flowchart

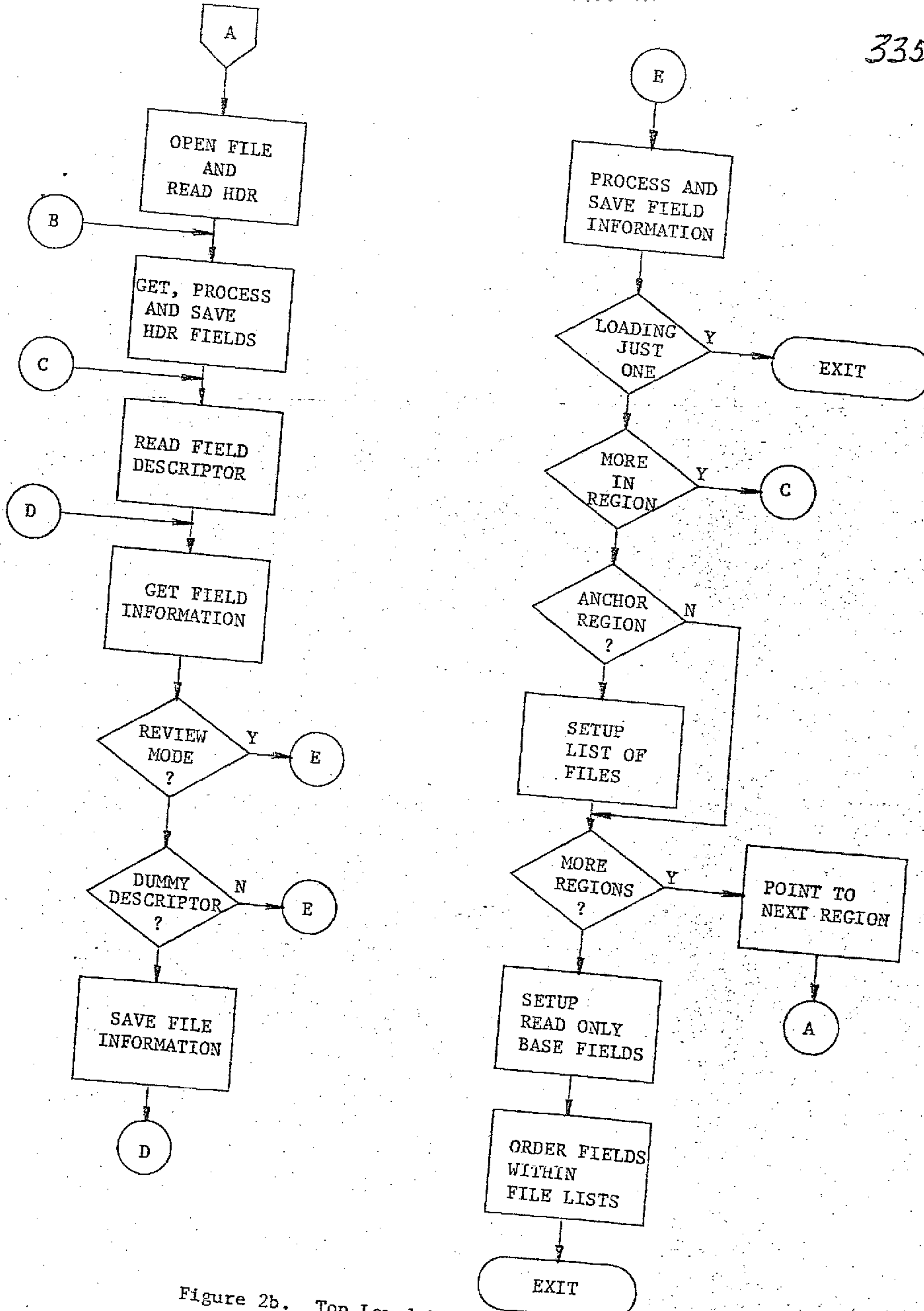


Figure 2b. Top Level Flowchart

TOPIC D.22 - DESCRIPTOR EDITOR - MOVE COMMAND

A. MODULE NAME

Program-ID - NDBEDMO
Module-ID - DBEDMO

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The MOVE command permits the user to reorder fields within any field list.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDMO:

1. FLD
2. HDR
3. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The user is prompted for the new position field name. If the entered field name does not exist, the user is given a diagnostic and prompted for a new fieldname. The new position field name cannot be, the anchor key field if the anchor file has record security, the subfile parent key field if the subfile has record security or the subfile key field, or the RECLEN field. If any of these conditions are met, the user is given a diagnostic and reprompted for a new position fieldname. A superfield has no field position. If a subfield is specified, the defining base field is located and used as the new position fieldname. All other fields are unacceptable.

The user is prompted for the field to be moved. To be valid, the field must exist and must not be a reserved fieldname, must appear in the same field list as the new position field name and must not be a superfield or a subfield. If the field is invalid, the user is given a diagnostic and reprompted for the field to be moved.

The field to be moved is deccupled from the list by resetting the appropriate forward and backward pointers. It is then threaded into its new position by setting the appropriate forward and backward pointers.

The command string is saved in the current strategy and then control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

   PL/I with TSPL/I statements.

2. Suggestions and Techniques

   Not Applicable

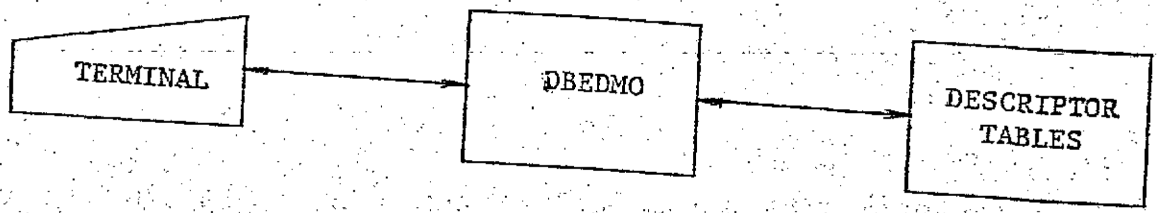


Figure 1. I/O Block diagram

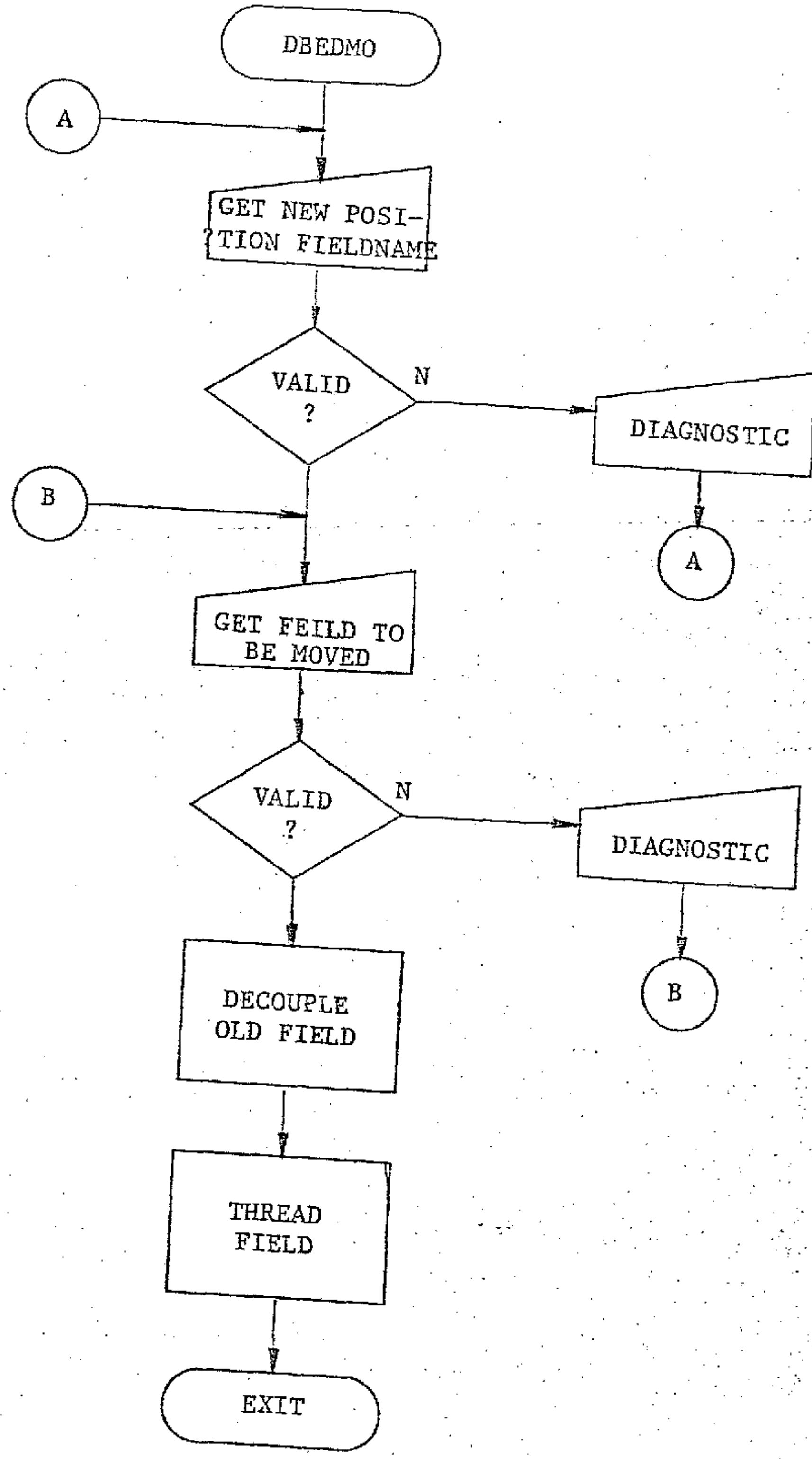


Figure 2. Top level flowchart

TOPIC D.23 - DESCRIPTOR EDITOR - PATCH COMMAND

A. MODULE NAME

Program-ID - NDBEDPA
Module-ID - DBEDPA

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The Patch command permits the user to patch the value in any descriptor record in any description region in the descriptor file. The record to be patched must be identified by use of the REVIEW command.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDPA:

1. FLD
2. HDR
3. RECSEC
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The REVIEW command indicates in X that a REVIEW has been done and it is alright to patch, REVIEW also indicates whether the field to be patched is a field or header descriptor. If a REVIEW has not been done the user is given a diagnostic and control is returned to the calling routine.

The user is prompted for his patch in the form "keyword=test". The keyword is checked to see if valid. If not, the patch is ignored, the user given a diagnostic and reprompted for the patch. If the name is valid, a branch is taken to the piece of coded which actually posts the appropriate field.

In each of the sections of code, one for each descriptor field name, a resonableness check is made on the patch text, to assure that the data will be accepted by the validation routines when posting the information to the descriptor file.

Refer to the Descriptor Editor Users Guide for the acceptable range and form of the patch texts.

The user may enter a parenthesesed list of patches.

After all the patches have been posted in the descriptor table work areas, they must then be

posted to the descriptor data set. The routine DBEDFD3 is called to post the appropriate field descriptor, or the routine DBEDFI is called to post the apprcpriate header descriptor. The routine called depends on whether the user is patching a field descriptor or a header descriptor. Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

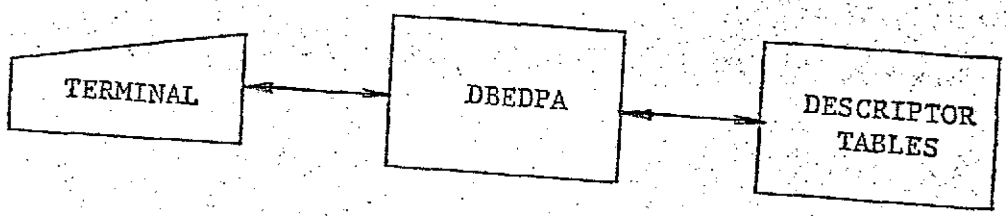


Figure 1. I/O Block diagram

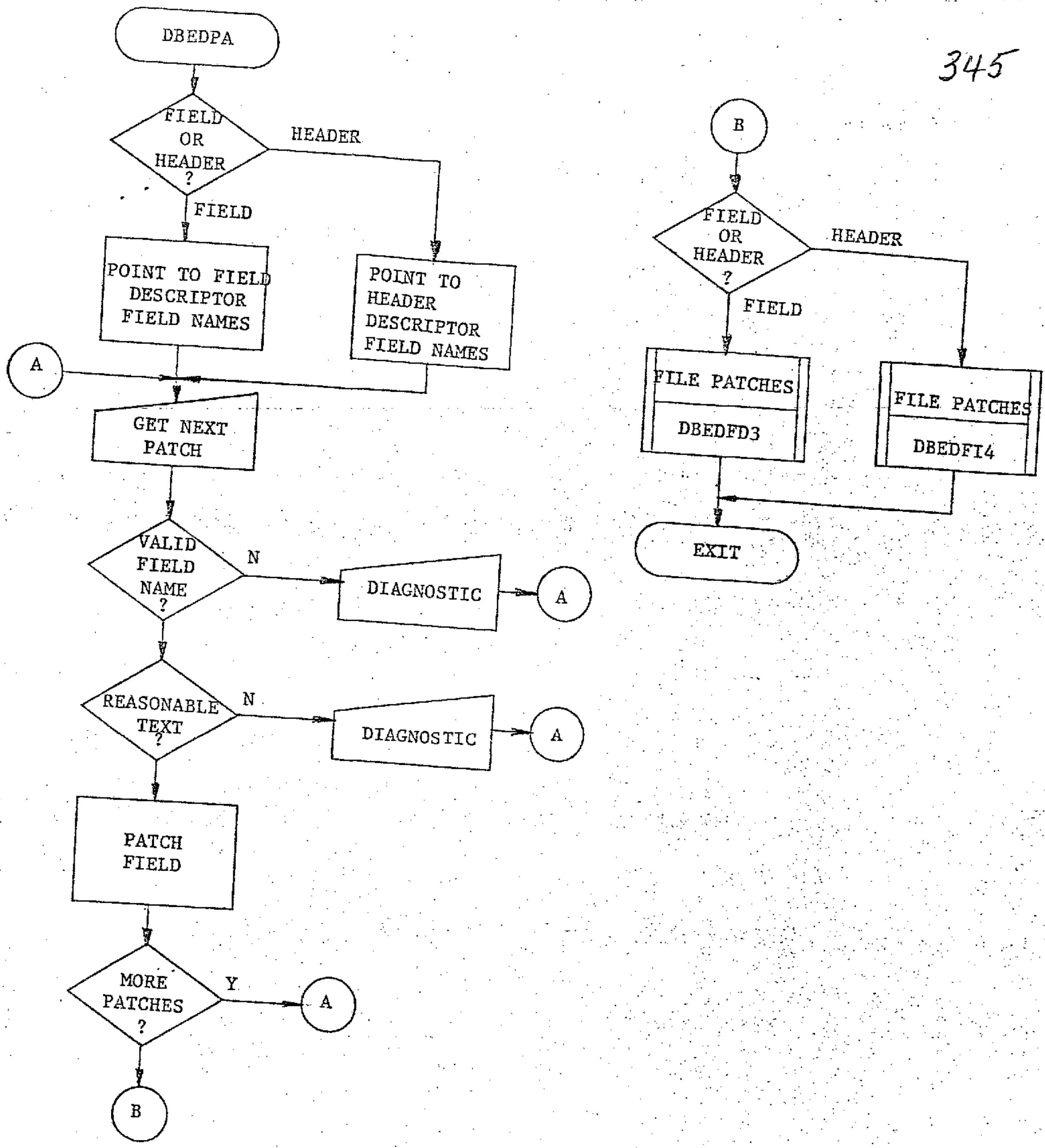


Figure 2. Top level flowchart

TOPIC D.24 - DESCRIPTOR EDITOR - PRINT COMMAND

A. MODULE NAME

Program-ID - NDBEDPR
Module-ID -DBEDPR

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The PRINT command gives the user a formatted printout of the descriptor information as it exists in core at the time the print is issued.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

The output data from NDBEDPR is placed in the SAM data set LIST.DESC from where it is printed on the high speed printer by an OS job. For the details of the data set refer to the dataset specifications.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

The informaticn stored in LIST.DESC is printed using column cne of each record as a carriage ccntrcl.

4. Reference Tables

The following external tatles are referenced by NDBEDPR:

1. FIELD
2. FLD
3. HDR
4. RECSEC
5. SECURITY
6. SUPER
7. VALID
8. X

A descripticn of these tables can be found dataset specificaticns in the of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The data set LIST.DESC is created using the assembler routines. A DCB is created for the output file by the routine ASMDCB, the JFCB set up by the routine ASMFNDS, and the dataset opened by the routine ASMOPEN.

The title lines for the data tase name are output by the routine ASMPUT. The data base name is output followed by two trailing title lines.

The title lines fcr the field descriptors are output. The lines of field information for each field are built and output.

After the field information is processed, the title lines for the header descriptor information are written out. The lines of header information for each descriptor region are built and written out.

The LIST.DESC dataset is closed by calling the routine ASMCLOS.

F. CODING SPECIFICATIONS

1. Source Language

   PL/I with TSPL/I statements.

2. Suggestions and Techniques

   Not Applicable

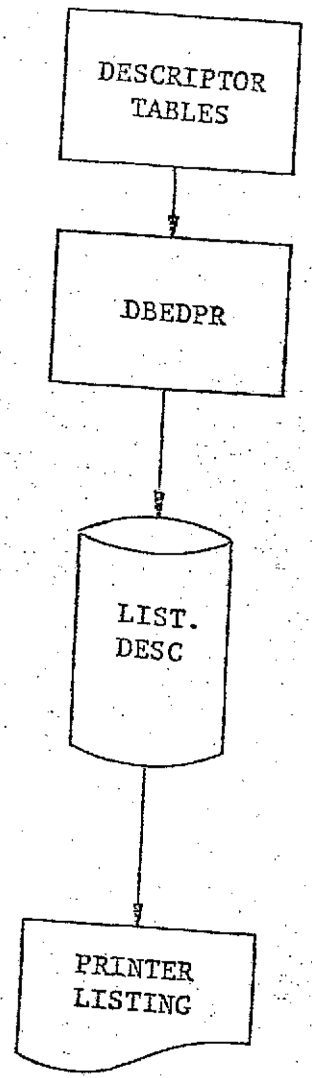


Figure 1. I/O Block diagram

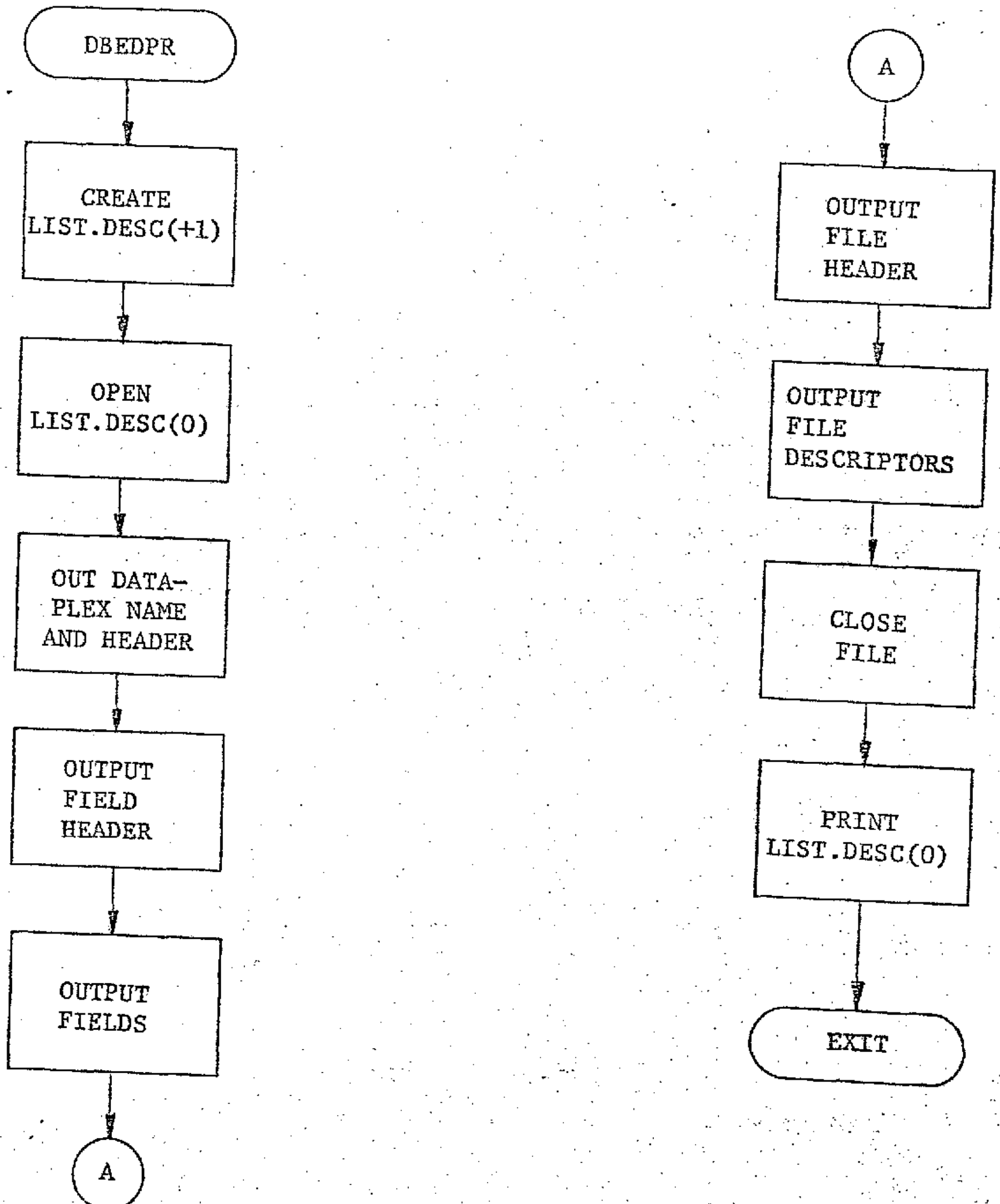


Figure 2. Top level flowchart

TOPIC D.25 - DESCRIPTOR EDITOR - RECORD SECURITY COMMAND

A. MODULE NAME

Program-ID - NDBEDRS
Module-ID - DBEBDRS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to create and define record security for any data base file except for indicies.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDRS:

1. FIELD
2. FLD
3. FLD_STRING
4. MDR
5. RECSEC
6. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   Routine DBEDGF is called to obtain a fieldname used to define on which file record security is to be placed. If in update mode and the header record is not loaded, DBEDLD is called to load the header. If there is no record security currently defined for the file in UPDATE mode, the user is given a diagnostic and control is returned to the calling routine.

   The user is prompted for a record security code. The add-delete indicator is removed from the code and validated. If it is invalid, the security code is rejected, the user is given a diagnostic and reprompted for the security coded. If no indicator is entered, "ADD" is assumed.

   The security code is removed from the parameter. If this is not an alphanumeric character string, the security parameter is rejected, the user is given a diagnostic and reprompted for the security code.

   The security mask to be valid must be a two digit hexadecimal character string. If it is invalid, the security parameter is rejected, the user is given a diagnostic and reprompted for the security parameter.

   Once the security parameter is validated, it is saved in an internal work area. The user may enter a list of security parameters as a list in parentheses. Each security parameter is obtained

from the user and processed as above.

If record security has been previously defined for the file, a pointer is set up to the file header and record security information. Otherwise a record security field is created and placed in the appropriate position in the fixed field list of the file. A record security save area is allocated and initialized.

A control loop is set up to process each entered security code. The existing security list if any is searched for the entered security code. If the security code exists and the new code is to be added, the two security masks are logically OR'ed together and the result posted in record security structure. If the code is to be deleted, the two security mask are logically exclusively OR'ed and the result placed in the record security structure. If the security code is not in the existing list and it is to be added, it is placed at the end of the existing list. If the code to be deleted and it does not appear in the list, it is ignored. Each security code is processed in this manner.

After all security code have been processed and the record security list is empty, the area is released and the record security field deleted from the file.

If in UPDATE routine DBEDFI is called to post the record security tc the descriptor file. The command string is saved in the current strategy and then control is returned to the calling module.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techninques

Not Applicable

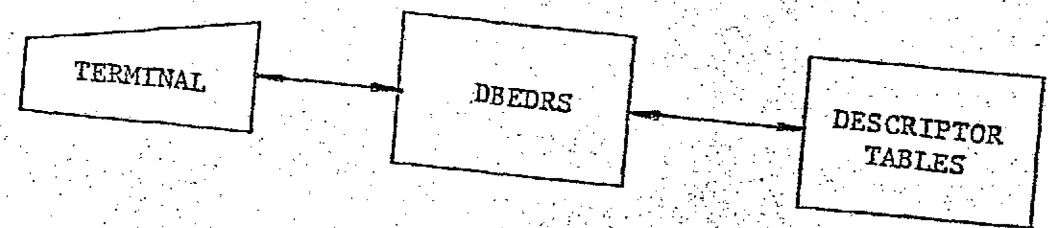


Figure 1. I/O Block diagram

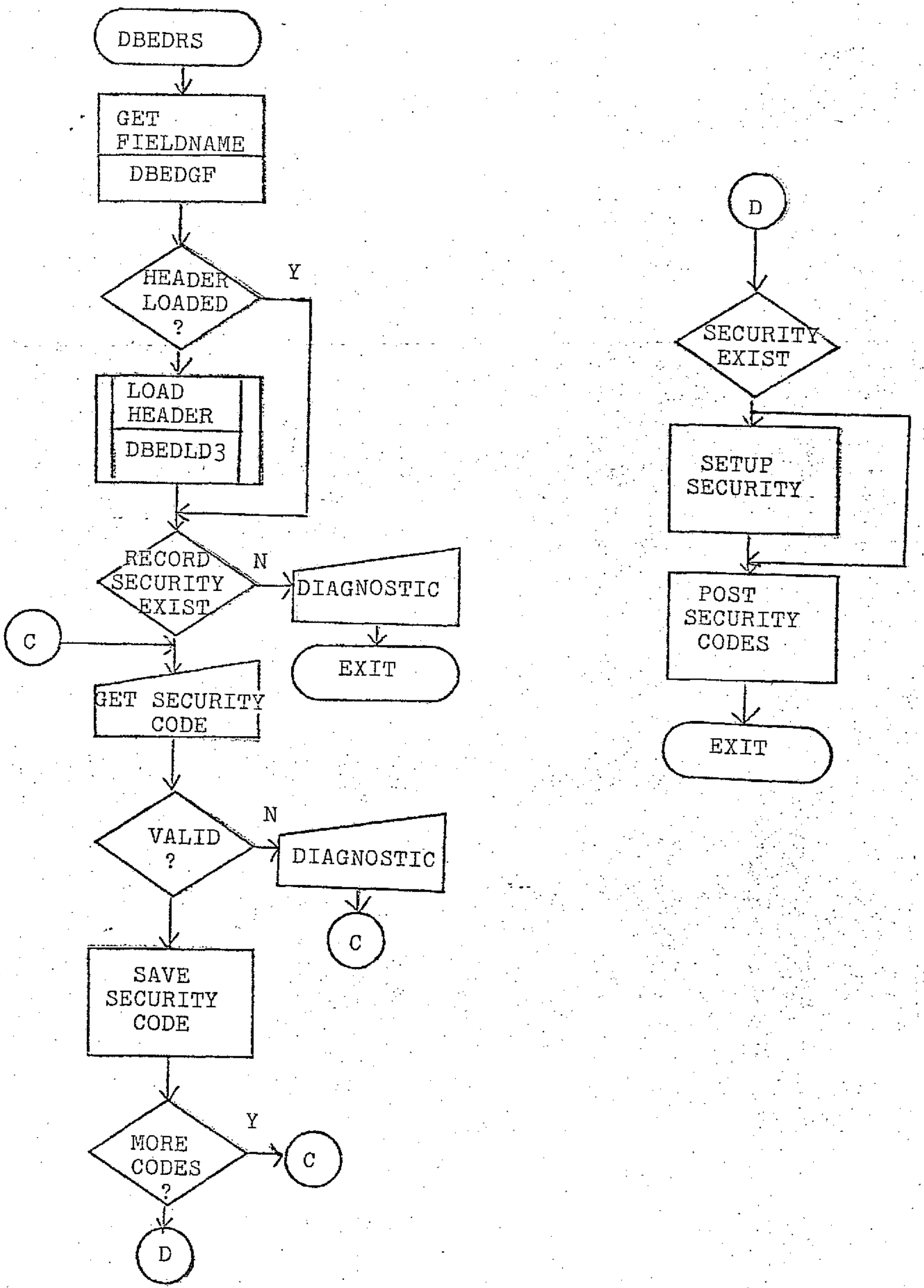


Figure 2. Top level flowchart

TOPICS D.26 - DESCRIPTOR EDITOR - RESTORE COMMAND

A. MODULE NAME

Program-ID - NDBEDRT
Module-ID - DEEDRT

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to restore the descriptor tables from a SAM data set to memory, so that the user may continue to create and/or modify the descriptors from their point of existence at the time the checkpoint was issued.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The input file is a SAM data set named DESCRP.CHKPOINT. Refer to the dataset specifications for a description of this data set.

3. Output Data Sets

a. Output Files

Not Applicable

b. On Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEDRT:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC_STR
7. SECURITY_STR
8. SUPER_STR
9. VALID
10. X

The description of these tables is in the specifications of the dataset DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into NDBEDRT a DCB is set up for the dataset DESCRP.CHKPOINT by calling the routine ASMDCB. ASMFNDS is called to setup the JFCB. Any and all existing field descriptors and file descriptors are released and then the FIELD structure itself is released.

ASMOPEN is called to open the dataset. That part of the X structure which was saved is read in over top of the same part of the existing X strucure.

The FIELD structure is allocated next. Note that the variable defining the size of FIELD structure is in that part of X which has just been restored. The FIELD structure is read in overlaying the just allocated existing FIELD structure.

A field descriptor is read into a work area. A FLD structure is allocated on the field information moved into it. If the FLD has field security, a validation argument, or is a super field, the appropriate structures are allocated,

the information moved into them, and the pointers in FLD setup accordingly. The changed flags in FLD are setup so that all of the field descriptor information will be forced out to disc when the FILE command is issued. Each field descriptor is processed in this manner.

A file descriptor is read into a work area. An HDR structure is allocated and the header information moved into it. If the file has record security, a RECSEC structure is allocated, the information moved into it, and the pointer posted into the HDR structure. The HDR pointer is posted into the proper slot in X.HEAD_TAB. Each header descriptor is processed in this manner.

The dataset DESCRP.CHKPOINT is then closed and control is returned to the calling program.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

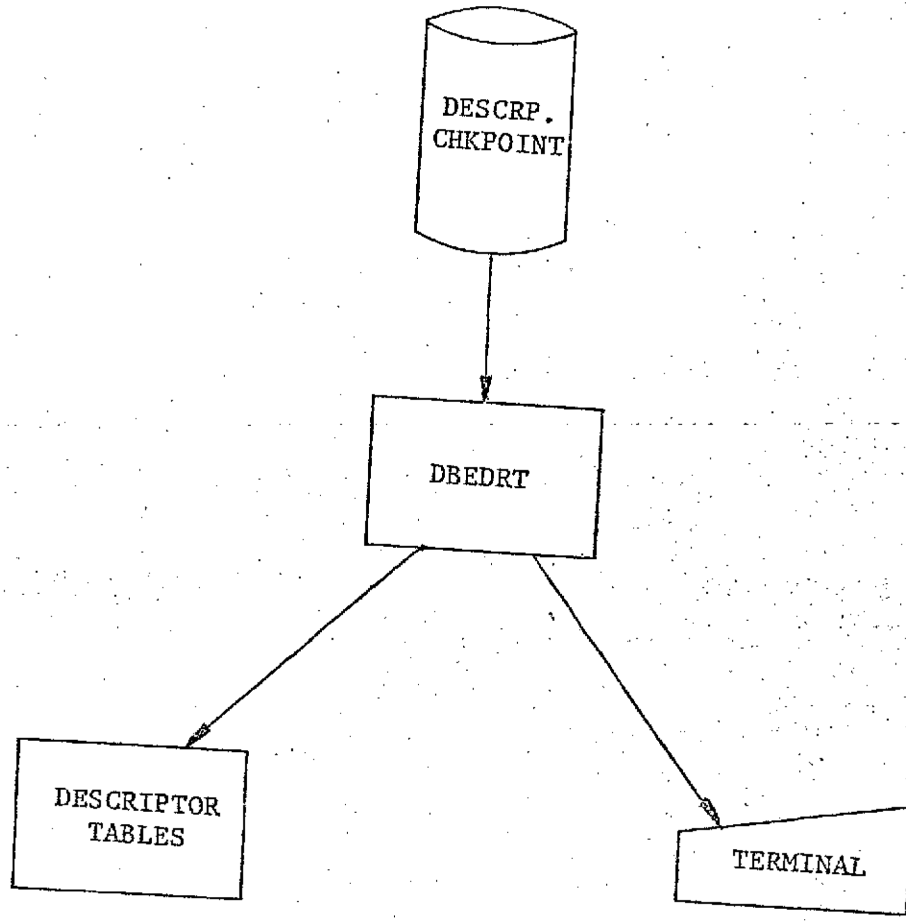


Figure 1. I/O Block diagram

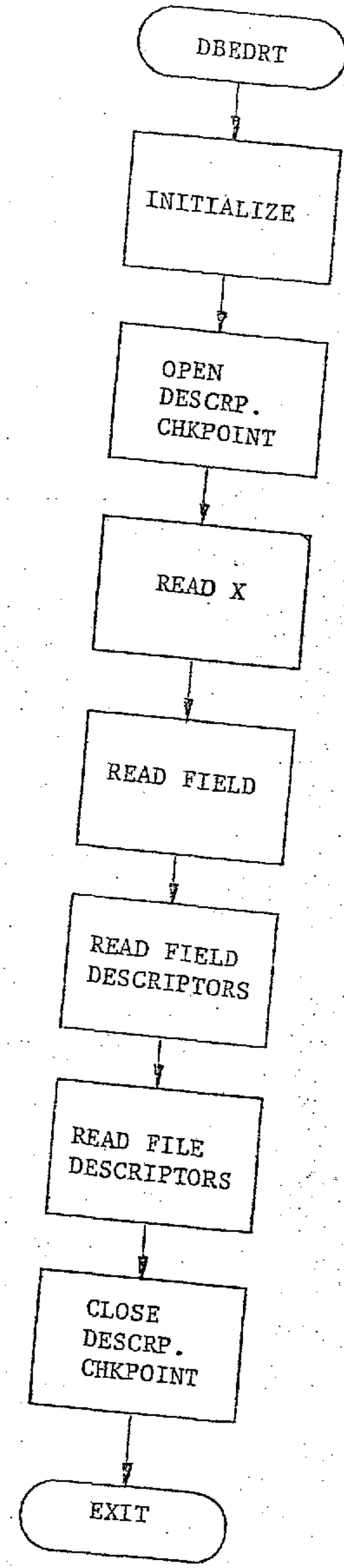


Figure 2. Top level flowchart

TOPIC D.27 - DESCRIPTOR EDITOR - REVIEW COMMAND

A. MODULE NAME

Program-ID - NDBEDRV
Module-ID - DBEDRV

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to present the descriptor information to the user of any descriptor record in any descriptor region in the descriptor file. Review points to the record to be patched by means of the PATCH command.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input File

      Not Applicable

   c. Input files

      The data base descriptor file is a ISAM file maintained by DBPAC, containing the information defining and detailing the information contained in the data base.

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      The various pieces of information contained

in the descriptor record are displayed on the screen preceeded by the descriptor descriptor field name. All fixed fields are displayed within a 20 character string. The number of items per line for fixed field items is determined by dividing the screen width by 20. The varying elements are display one per line, with continuation lines if necessary.

c. Formatted Print-Outs

Not Applicable

4. References

The following external tatles are referenced by NDBEDRV:

1. FLD
2. HDR
3. RECSEC
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables is found in the dataset specifications of the dataset DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

At the command entry point the paging information structure is allocated and initialized. The user is prompted for the descriptor file a region id that he wishes to review from. If the region id is invalid, the user is given a diagnostic and prompted for a new region id.

The user is prompted for the name of the descriptor record he wishes to review from the descriptor region. If the descriptor name is invalid, the user is given a diagnostic and prompted for a new descriptor name. If the descriptor exists the routine DBEDLD is called to load the descriptor data. If a loading error occured, the user is given a diagnostic and prompted for a new descriptor value.

The paging information structure is setup to point to the first page of information to be displayed. At which point the command entry and paging entry join in common code.

At the paging entry point, the paging information is set to point to the proper page to be displayed and then join common code with the command entry point.

At the start of the common code, the number of the next item to be displayed is retrieved from the paging information and a branch is taken to the approprate code to obtain the next piece of descriptor information. Seperate pieces of code exist for each field descriptor and file descriptor descriptor fields. After the piece of information is built, it is inserted in the output line. If there is sufficient room in the output line for more data, the next item of information is obtained as above. If the line is full, it is put into the TS screen buffer.

If there are more lines of screen available, they are built and processed as above. This continues until either the screen buffer is full or all of the information has been exhausted. If the screen is full and there is more information to output in the forward direction, a paging entry point is setup and the next page of information is posted in the paging information. The buffer is then flashed to the screen.

The X structure is posted as the descripter region and field name of the record REVIEW'ed so that the user may use the PATCH command if he desires, after which control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I and DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

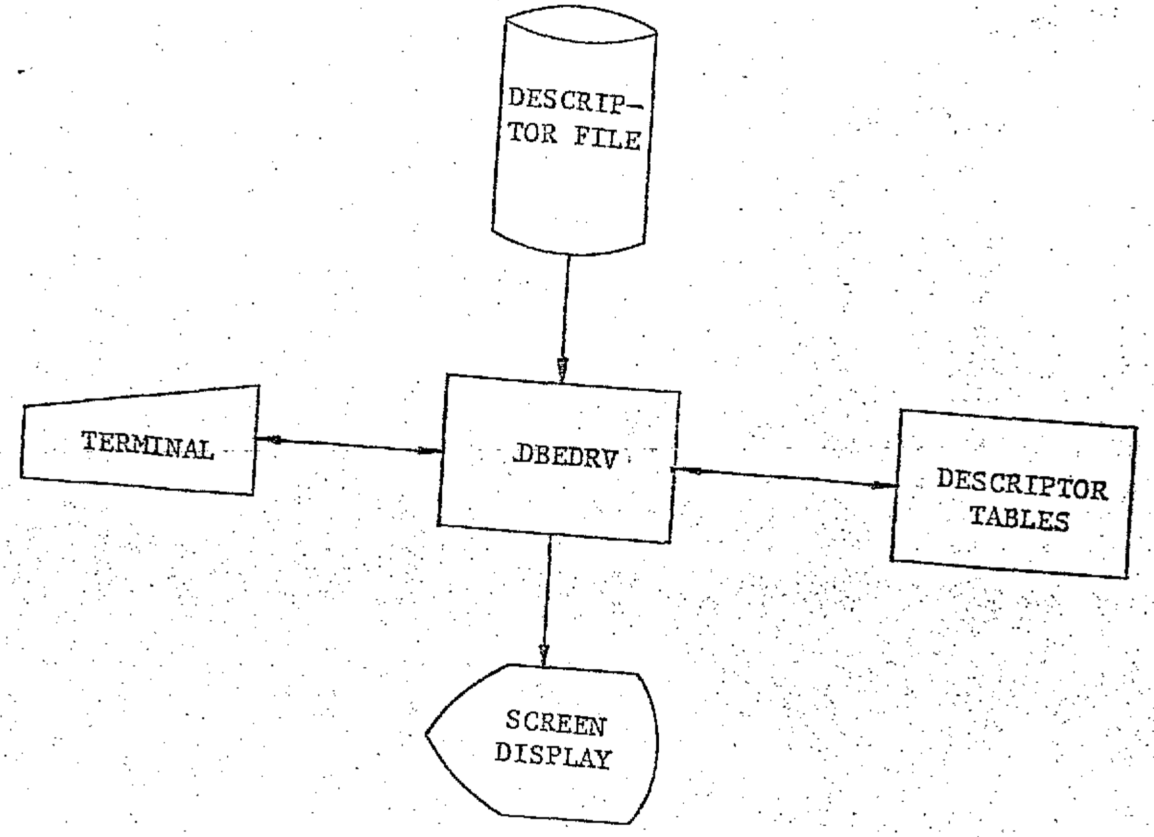


Figure 1. I/O Block diagram

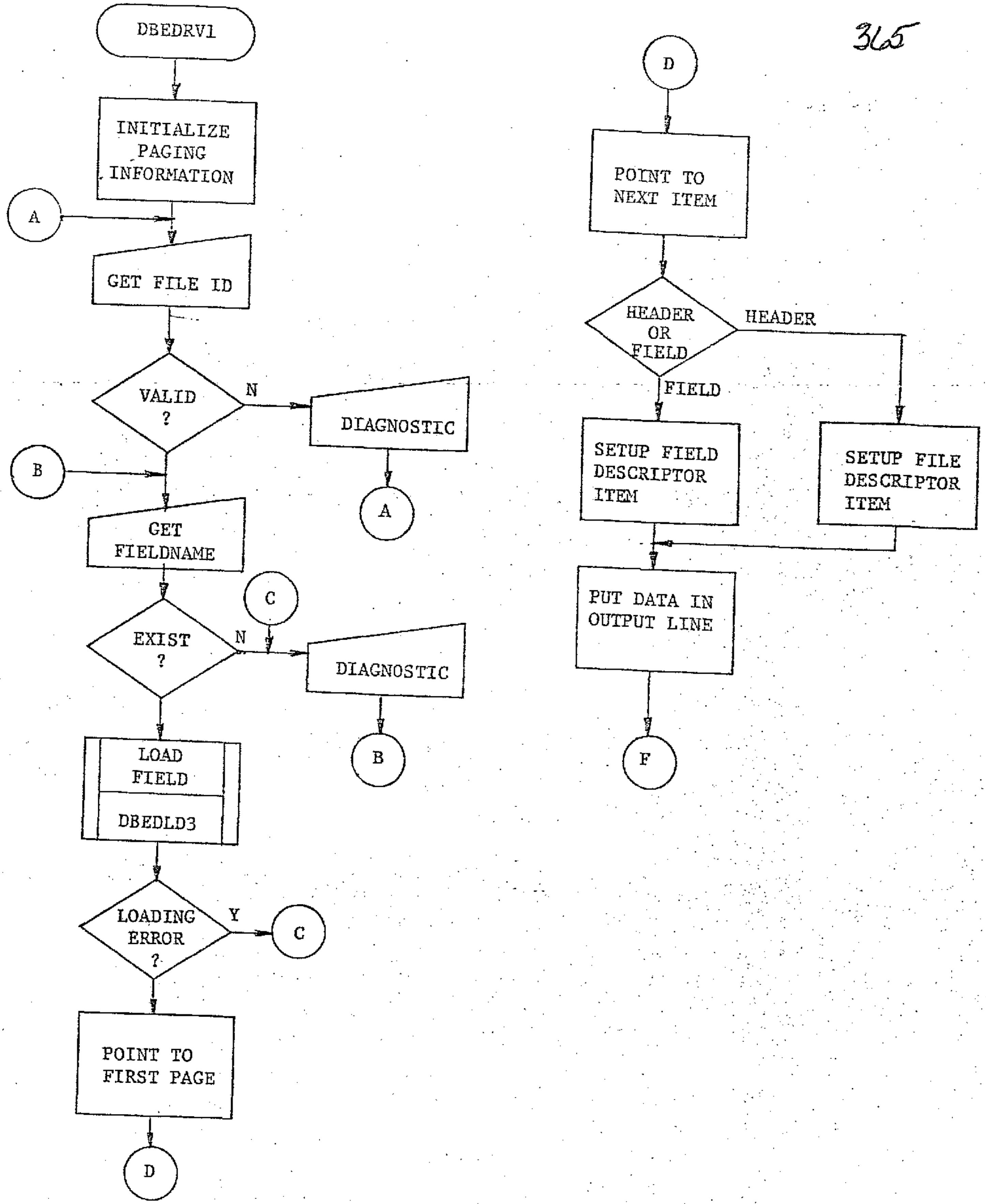


Figure 2a. Top level flowchart

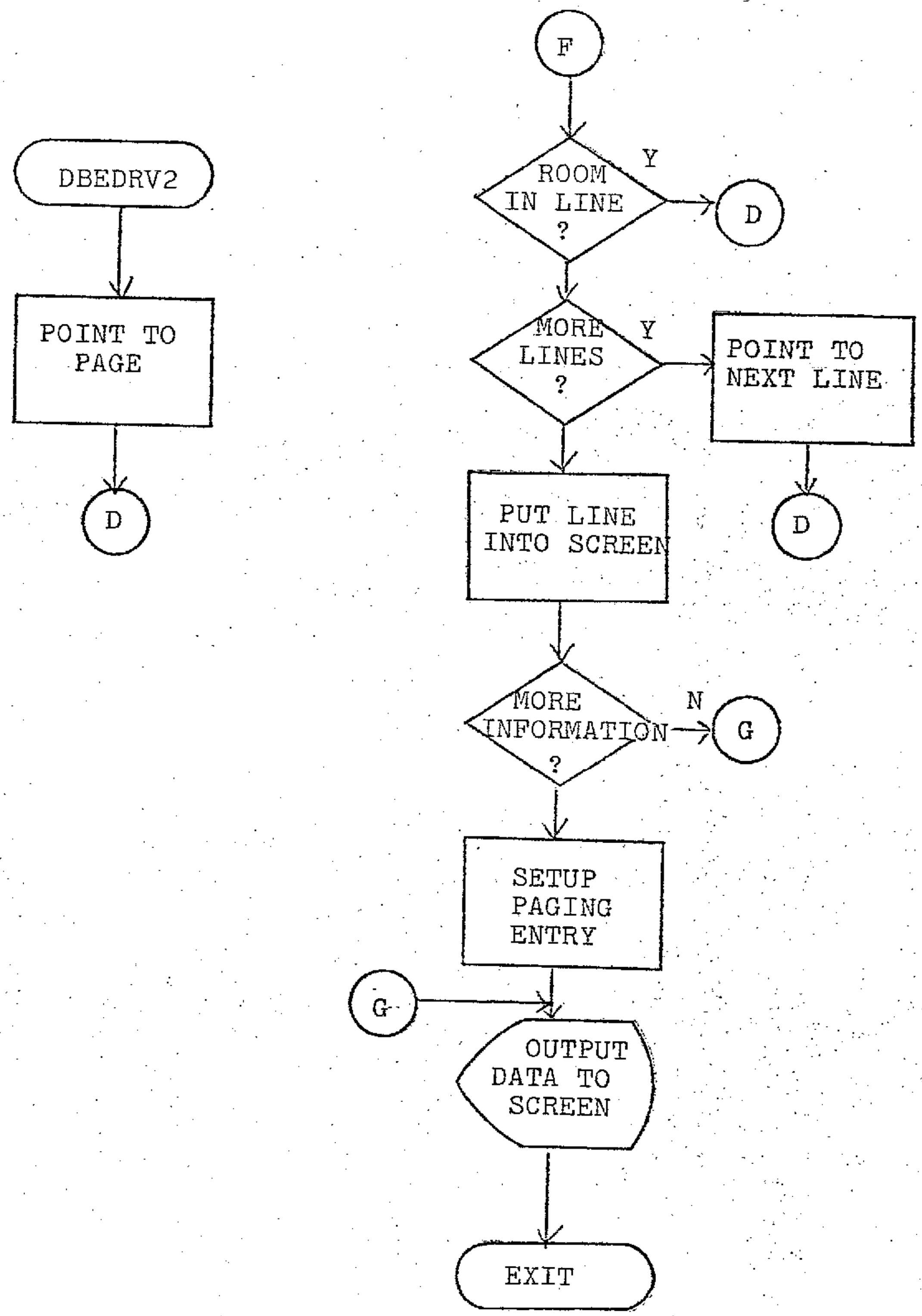


Figure 2b. Top level flowchart

TOPIC D.28 - DESCRIPTOR EDITOR - SAVE STRATEGY COMMAND

A. MODULE NAME

Program-ID - NDBEDSS
Module-ID - DBEDSS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The command is used to create and save in the strategy data set, a list of Descriptor Editor commands which when executed at any future time will create a set of descriptors exactly like those that exist in core at the time the SAVSTRT command is issued.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-Line Terminal Displays

      Not Applicable

   c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by NDBEPSS:

1. FLD
2. HDR
3. RECSEC
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into SAVSTRT, the user is prompted for the strategy name in which the Descriptor Editor commands are to be saved. If the name is not of the proper form or a strategy by that name already exists, the user is given a diagnostic and prompted for a new strategy name.

Once a valid strategy name is obtained, the MAINTAIN and EDIT command strings are saved in the strategy. This initializes the strategy. The internal subroutine SAVE_FLD is called to save the ADD command to create the anchor file key field.

A control loop is set up to process each of the 20 possible files in the order of anchor file, associate file and then subfiles. With each existing file each field list is processed in the order of packed bit fields, fixed fields, and then varying fields. Each field list is processed from the start of the list to the end of the list.

The SAVE_FLD command is called for each field to create and save the appropriate command string.

The fields COMMENTS, FREEFORM, the subfile key field, and the subfile parent key fields are

skipped as they are created thru the adding of the anchor key field or the CREATSUB command. The record security field is skipped if encountered and processed after all other fields on the file have been processed. All the fields on all of the files are processed in the manner and order.

If the RECLEN field has field security, the SAVE_FS is called to build and save the FLDSEC command.

After all of the fields and files have been processed, the FILE and END commands are saved in the strategy, after which time control is returned to the calling routine.

In the SAVE_FLD internal procedure, if the field is a subfile control field the CREATSUB command string is built else of the field is a superfield the SUPERFLD command string is built, otherwise the ADP command string is built. The appropriate command string is saved thru use of the routine TSPUTG.

The internal entry SAVE_FS is defined at this point to save the field security if any. This code is also part of the SAVE_FLD procedure. If the field has field security defined on it, a FLDSEC command string is built or saved in the strategy through use of the routine TSPUTG. Control is then returned to the calling point in SAVSTRT.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

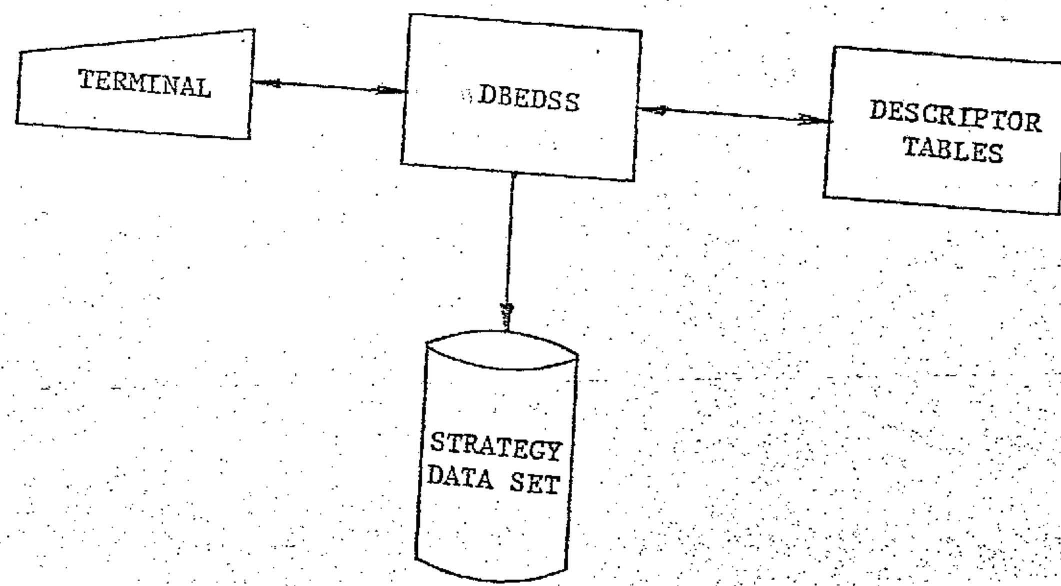


Figure 1. I/O Block diagram

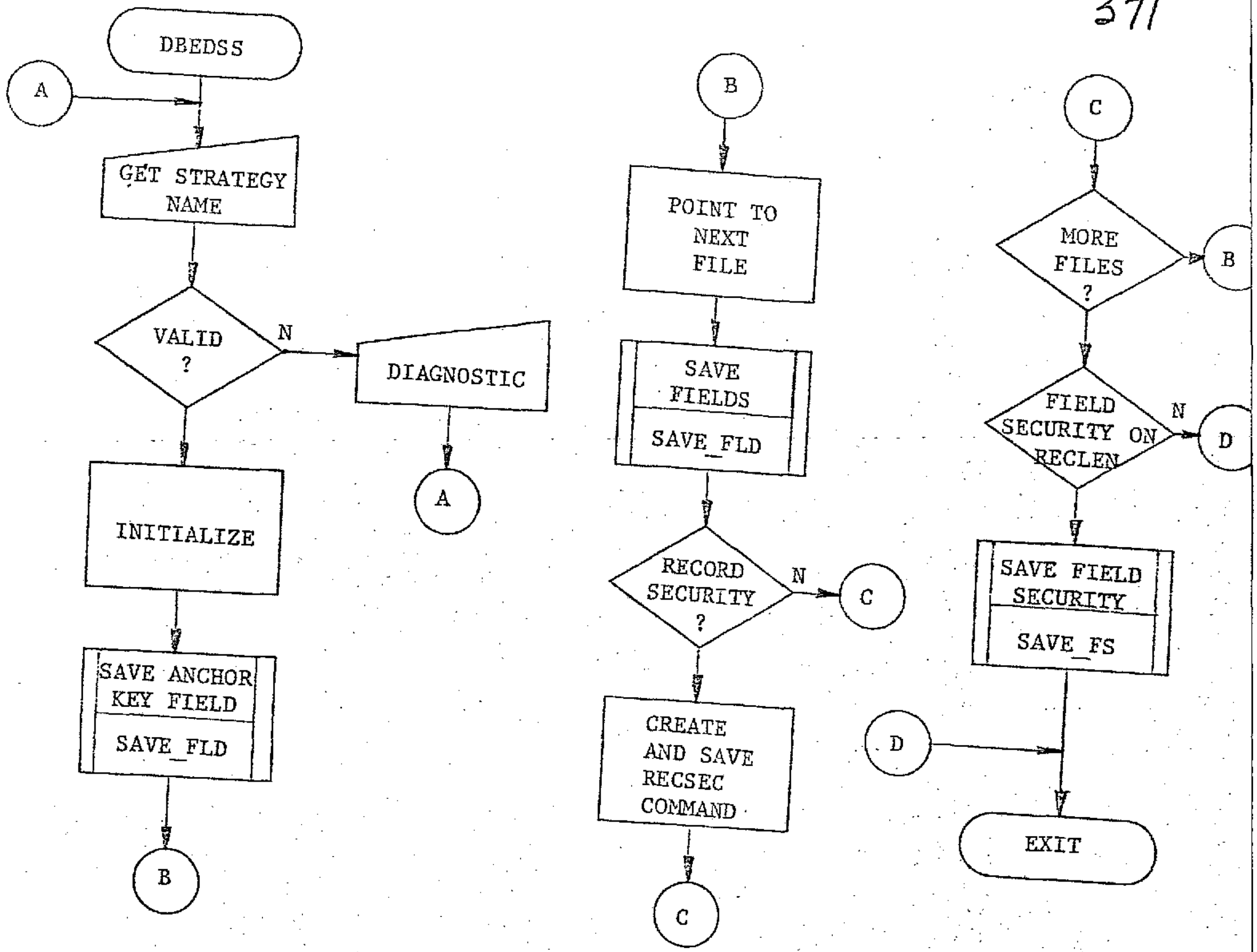


Figure 2a. Top level flowchart

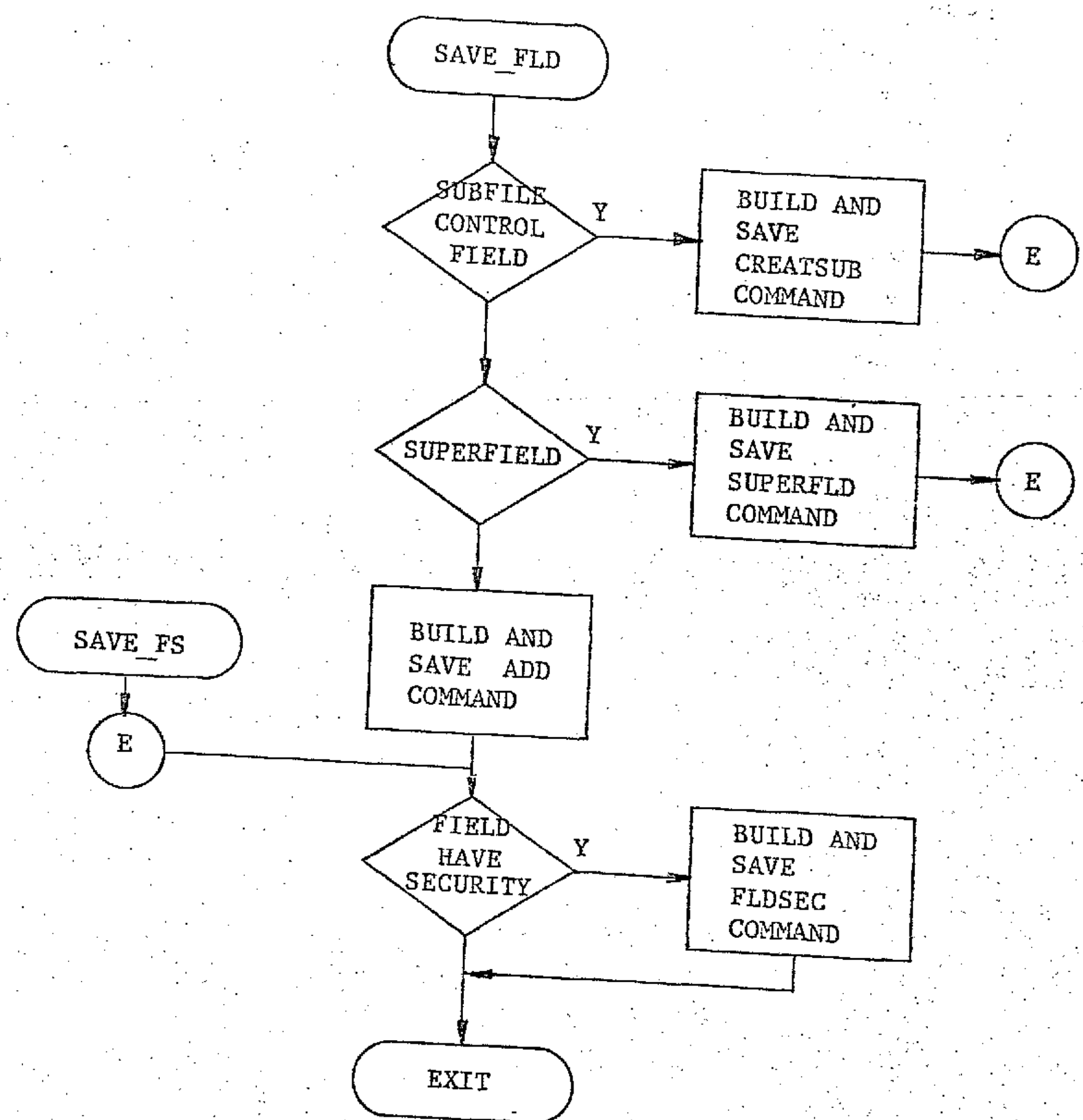


Figure 2b. Top level flowchart

TOPIC E.1 - TERMINAL SUPPORT - PREPROCESSOR

A. MODULE NAME

Terminal Support PL/I Preprocessor
Program-ID - TS
Module-ID - TS

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

TS analyzes terminal input/output PL/I language extension statements and produces statements acceptable to the PL/I compiler. These statements call the terminal support module allowing the program to communicate with the user's SYSIN and SYSOUT or, pending TSS support, an on-line display station. The user's SYSIN and SYSOUT are a terminal if the task is conversational, or data sets, if non-conversational. Diagnostic messages are generated for errors which can be detected by TSPL/I during preprocessing.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      The Job Control cards needed to invoke the PL/I compiler for OS are described in the IBM PL/I Programmer's Guide.

   b. Punched Card Input Files

      1. TS Text

         The TS text deck is all text for insertion into the source program following a "% INCLUDE LISRMAC(TS);" statement in the source program. This text consists of the source statements of the TS preprocessor function and any PL/I statements to be inserted at the "%

INCLUDE IISRMAC(TS);" statement in the source program. The TS text is coded as specified in this report, formatted according to PL/I source language standards, and catalogued in a data set for compile time use by all programs using TS.

2. Source Deck

The source deck is any PL/I source program using TS statements to interface with the user's SYSIN and SYSOUT or any on-line display station. The statement formats and their use are described in the TSPL/I User's Manual (Section II, Topic E.2 of the DWB).

c. Input Files

The TS text statements are catalogued as a member of a partitioned direct access data set for retrieval by the IBM PL/I precompiler. This data set is accessed via ddname LISRMAC.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The object module consists of the relocatable machine instructions and constants generated by the PL/I compiler for the source program. It is stored in a partitioned data set. This data set is that one defined in the compiler job step with the DDNAME SYSLMOD. The module is linked by the OS linkage editor.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

1. Precompiler Listings

Two precompiler listings are produced:

(a) A source listing before

precompilation and

(b) Any precompiler diagnostics (i.e., errors in the use of preprocessor PL/I, not TS error messages). The IBM PL/I Programmer's Guide explains the listing format.

2. Compiler Listings

The compiler listings produced include an intermediate source listing (between precompiling and compiling) and any compiler diagnostics. Any errors detected by the precompiler function will generate PL/I comments in the intermediate source listing. Serious TS PL/I errors may result in compiler diagnostics also. The IBM PL/I Programmer's Guide explains the listing formats.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. TC - terminal control block

b. TSPL/I - diagnostic comments.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Top Level

The mainline PL/I source program is required to have a "% INCLUDE LISRMAC(TS);" statement once in the program before all TS preprocessor function references. This statement directs the PL/I precompiler to take text from member TS of the library accessed via ddname LISRMAC and incorporate it into the source program. (Refer to the TS block diagram in Section D.1 of this write-up).

The TS function receives one argument from a preprocessor function reference; i.e., a variable length character string. It is TS's function to scan and parse this input string to determine if it is in the correct format and then to generate a string called the "generated text." This string consists of valid PL/I statements and comments for communication with the terminal support modules.

The processing of TSPI/I is closely analogous to the processing of DBPL/I described in Section IV, Topic A of the DWB and is only summarized here. The TS text declares and activates the TS preprocessor function. Argument initialization, finding a subargument, passing labels and comments through, and finding the statement keyword to select the specific statement routine are all done analogously to the DBPL/I preprocessor function. Diagnostic comments are generated for any errors detected. (See Section III, Topic E.1 of the DWB.) There are no files to be analyzed.

In all programs a declaration of the entry point to the terminal support modules is generated and a declaration of TC - the Terminal Control block (See Section III, Topic E.2 of the DWB.)

b. Specific Statement Routines

Each specific statement routine examines the statement from left to right until the semicolon clause is found. The keywords are verified for correct spelling and order. If any error is detected, a diagnostic comment is generated and the statement abandoned by control being transferred to the inter-statement point. Following successful analysis, each specific statement routine generates PL/I statements for communication with the terminal support modules and loops back to the inter-statement point.

The ON PAGE statement routine generates the following statement:

```
TC.PAGING_ENTRY=expression;
```

Where "expression" is taken from the CALL

clause of the TS ON PAGE statement.

The ENTRY statement routine generates the following statements:

```
TS_ENTRY_RETURN_POINT=TS_ENTRY_LABEL_n;
GO TO TS_ENTRY_CODING;
TS_ENTRY_LABEL_n:
```

Where "n" is a numeric value assigned sequentially to each ENTRY statement as it is encountered.

The ENABLE statement routine generates the following statements:

```
DCL TS_ENTRY_RETURN_POINT LABEL;
TS_ENTRY_RETURN_POINT=TS_ENTRY_LABEL_1;
TS_ENTRY_CODING:
ON CONDITION(END)
   GO TO TS_EXIT_CODING;
ON CONDITION (ATTN);
TC.FUNCTION='ENTRY';
CALL TSCNTRL(TC);
GO TO TS_ENTRY_RETURN_POINT;
TS_EXIT_CODING;
RETURN;
TS_ENTRY_LABEL_1:
```

Lines 4-6 and 10-11 of the above text are only generated when the user specifies the appropriate option on the ENABLE statement.

The TS logic is such that the ENABLE statement, if it appears, must appear before the first ENTRY statement, and in fact, implies an ENTRY statement. Likewise, the first ENTRY statement implies a default ENABLE statement, if none are present.

The PROMPT statement routine generates the following statements:

```
TC.FUNCTION='PROMPT-e';
TC.PROMPT.MESSAGE_KEY=expression;
TC.PROMPT.KEYWORD=value;
CALL TSPRMTe(TC,variable,list);
```

Where "expression" is taken from the MSG clause of the statement, "value" is taken from the KEYWORD clause (if present), "variable" is taken from the INTO clause (if present) and "list" is taken from the USING

clause (if present). The value of "e" is generated according to the following table:

1. INTO clause - none
   "e"=M

2. KEYWORD clause - none
   "e"=C

3. KEYWORD clause - yes
   "e"=D

The READ statement routine generates the following statements:

```
TC.FUNCTION='READ';
CALL TSREAD(TC,variable);
```

Where "variable" is taken from the INTO clause of the TS READ statement.

The WRITE statement routine generates the following statements:

```
TC.FUNCTION='WRITE';
CALL TSWRITE(TC,variable);
```

Where "variable" is taken from the FROM clause of the TS WRITE statement.

The PUT statement routine generates the following statements:

```
TC.FUNCTION='PUT';
TC.OUTPUT.POSITION='a';
TC.OUTPUT.DIRECTION='b';
CALL TSPUT(TC,variable,value);
```

Where "variable" is taken from the FROM clause of the TS PUT statement and "value" is taken from the TAG clause (if present). The value for "a" will be generated according to the following table:

1. position clause - none
   "a"=0

2. position clause -LINE
   "a"=0

3. position clause - PAGE
   "a"=1

The value for "b" will be generated according to the following table:

1. direction clause - none
   "b"=0

2. direction clause - FORWARD
   "b"=0

3. direction clause - BACKWARD
   "b"=1

The FLUSH statement routine generates the following statements:

```
TC.FUNCTION='FLUSH';
CALL TSFLUSH(TC);
```

The FINISH routine sets a precompiler variable to indicate that a FINISH statement has been processed and to prevent the processing of any further TSPL/I statements. A diagnostic comment indicating the number of TSPL/I errors is generated. If there have been any errors detected, the following statement will be generated causing an IEM0512I PL/I error:

```
DCL TS_DUMMY_VARIABLE LABEL
INIT(TS_ERRS_nn);
```

Where "nn" is the number of TSPL/I errors detected.

All statements and comments generated will be aligned as seventy-one byte strings, for ease of analysis.

F. CODING SPECIFICATIONS

1. Source Language

TS is written in IBM PL/I preprocessor statements.

2. Suggestions and Techniques

Not Applicable.

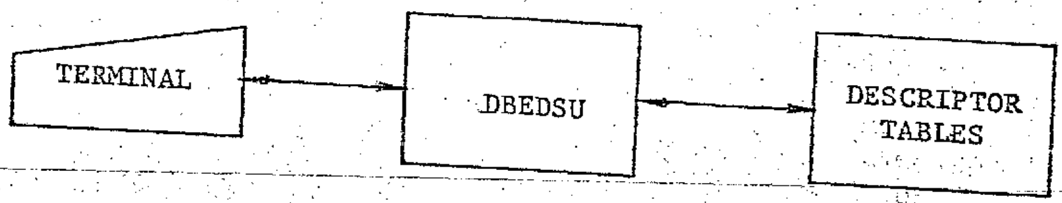


Figure 1. I/O Block diagram

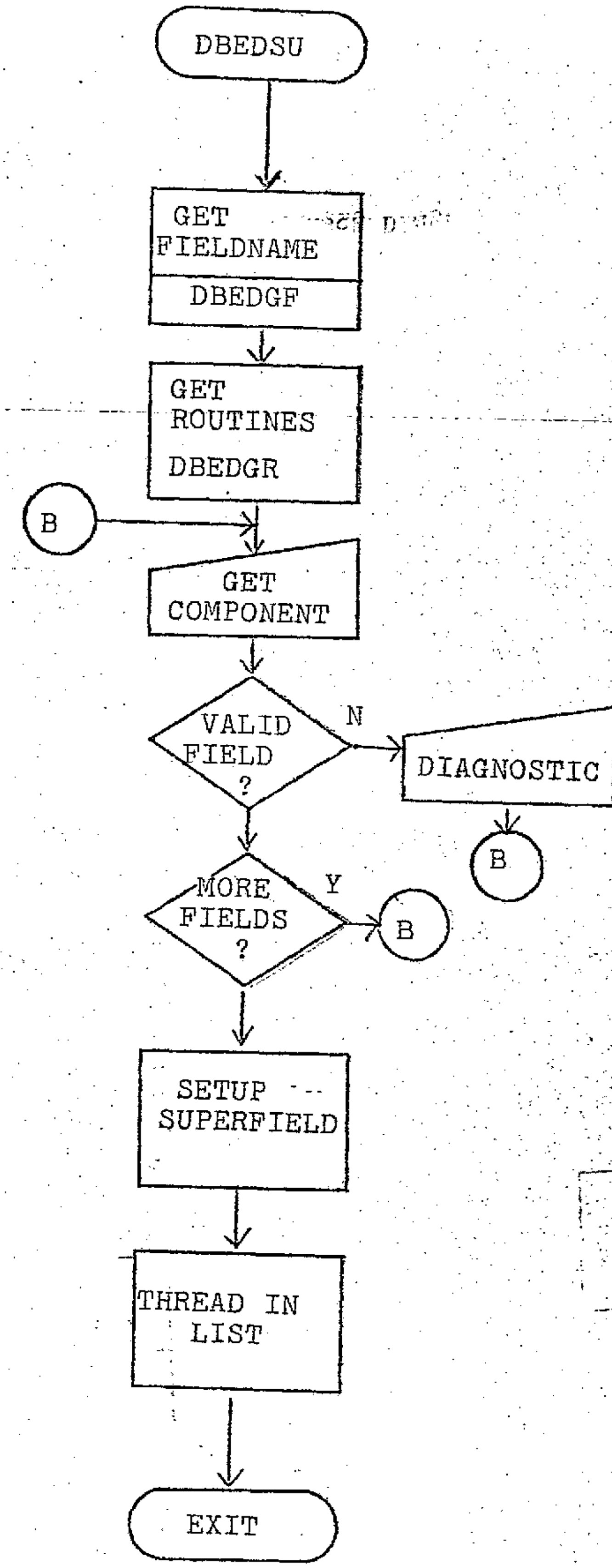


Figure 2. Top level flowchart

TOPIC E.2 - TERMINAL SUPPORT SUPERVISOR

A. MODULE NAME

Terminal Support - Terminal Support Supervisor
Program-ID - NTSUPER
Module-ID - TSUPER
Entry Points - TSATIN, TSCNTRL, TSFLUSH, TSGETKY, TSPRMTC, TSPRMTD, TSPRMTM, TSPUT, TSREAD, TSWRITE

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

TSUPER is the primary vehicle of communications between the NASIS monitor (MTT or stand-alone) and the NASIS PL/I data Base programs. Among the functions TSUPER performs are:

a. Issues I/O requests from data base programs. This includes command, data and message prompts and ordinary read and write requests.

b. Initializes the Terminal Control Block (TC) for each PL/I program. Supplies information about the current display area dimensions and resets all bit switches to zero.

c. Controls asynchronous interrupt processing. Detects APOFF, END and GO conditions and insures that asynchronous activities do not interfere with normal processing.

d. Maintains a push-down stack of message key references to support the EXPLAIN facility.

e. Scans and passes user input strings for commands and data. Information entered at the terminal is interpreted and passed to requesting programs in useful segments. The TC Block is utilized to enhance interprogram communication.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 2

2. Input Data Sets

a. Parameter Cards

Not applicable

b. Punched Card Input Files

Not applicable

c. Input Files

1. NASIS Message file

d. On-line Terminal Entries

All responses to command and data prompts by NASIS programs pass through TSUPER. See the command System User's Guide for a detailed discussion of the format and syntactic rules for these responses.

3. Output Data Sets

a. Output Files

Not applicable

b. On-line Terminal Displays

All output from NASIS data base programs passes through TSUPER. See the TSPUT and TSPROMPT sections of this TOPIC for a complete discussion of terminal output characteristics.

c. Formatted Print-Outs

Not applicable

d. Punched Card Output Files

Not applicable

4. Reference Tables

a. External Tables

1. TSCTL
2. USERTAB
3. TSCREEN
4. MTTUTAB

b. Internal Tables

1. EXPLIST

An area in which a push-down list of message keys is saved.

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

a. MAINLINE: See Figure 3

b. Entry Points:

1. TSATIN - See Figure 4

2. TSCNTRL - See Figure 5

3. TSFLUSH - See Figure 6

4. TSGETKY - See Figure 7

5. TSPRMTC - See Figure 8

6. TSPRMTD - See Figure 8

7. TSPRMTM - See Figure 9

8. TSPUT - See Figure 10

9. TSWRITE - See Figure 11

c. Program Subroutines:

1. GETPR - See Figure 13

2. DELPR - See Figure 14

3. GETSYN and GETDFALT - See Figure 15

4. PULINEND - See Figure 16

5. SDGIVITD and SDGIVITC - See Figure 17

6. SDPASS and SDYSNCHK - See Figure 18

7. RESETBUF - See Figure 19

8. SDSTRIP and STRIP - See Figure 20

9. PRKEYSAV - See Figure 21

10. IMCHECK - See Figure 22

11. SIGNAL and SIGNALC - See Figure 23

12. SETLDAB - See Figure 24

13. GETMLF - See Figure 25

14. MOVE - See Figure 26

15. PROMPT - See Figure 28

16. EXIT - See Figure 29

2. Narrative

a. MAINLINE

All calls to TSUPER entry points pass through the MAINLINE code. The purpose of this code is to insure that each user has the correct work areas, to initialize base registers and to restrict TS usage during APOFF and ATTENTION processing.

Execution proceeds by calling the PLI service routine IHESADA to obtain a Dynamic Storage Area (DSA). Next, registers are initialized and useful pointers are saved in unique locations. The PLI Pseudo Register Vector (PRV) draws special attention since it is not maintained in register 12, as is the norm for other programs.

Using the PRV, MAINLINE determines if a copy of the TS PSECT has been allocated for this user. If not, the routine GETPR is invoked to obtain one. On return, data lifted from MTTUTAB is utilized to compute the user's logical and physical device dimensions and this information is saved for future reference.

The one-byte switch ICSW is checked to find out which entry point was entered. If entry was through TSATIN, control goes directly to the interrupt processing code. For any other entry, the contents of the user's TC Block (passed as a parameter) is moved to the DSA

for easy addressability. If an APOFF has been requested by the user, only calls to TSPRMTM and TSCNTRL are allowed to execute normally, all others being short circuited to the routine which signals an END condition. If not in APOFF mode, control is passed to the routine specified at entry.

b. Entry Points

1. TSATIN

This entry point is called by module TSATTN whenever it determines that a user attentions should be processed. If the user has previously entered APOFF, the attention is ignored. If an immediate command is currently processing, condition END is signaled which terminates the command. If this is the second successive attention and processing of the first is sufficiently advanced, condition END is signaled; otherwise, this interrupt is ignored.

For all first attentions not mentioned above, condition ATTN is signaled in the most recently activated ON CONDITION block. On return, a second copy of the user PSECT is allocated, the string input buffer is initialized to null input and the PL/I routine DBATTN is called to issue the ''-ATTN:'' prompt.

On return from DBATTN, all user requests have been satisfied and the user is ready to continue. After closing the duplicate DCBs for the message files, the duplicate user PSECT is released. If the user entered END or APOFF in response to ''-ATTN:'', then pointers are set to cause execution to resume at the PL/I signal routine for END condition; otherwise, execution resumes at the point of interrupt.

2. TSCNTRL

TSCNTRL is called by any program which anticipates calling TSUPER for input-output service. Its function is to initialize the TC Block for use and pass the user's terminal dimensions.

Terminal dimensions are obtained from the user's profile by repetitive calls to TSGDEF. If no defaults are specified, the necessary information is taken from MTTUTAB.

Control is returned to the caller through the EXIT routine.

3. TSFLUSH

TSFLUSH is the display output routine for terminal support. It is normally called after consecutive calls to TSPUT have caused an output buffer to be filled. If a buffer has overflowed and AUTOWRITE is indicated, this routine is called from TSPUT and a flag is set to cause the ''MORE:' message to replace the next prompt.

The name of the paging entry for the program doing a PUT or FLUSH should always be in the TC Block as it is saved by TSFLUSH just prior to the write. Data is output one line at a time for typewriters and in a block for screens. The most current display is saved in the external controlled storage named TSCREEN.

4. TSGETKY

This entry point is called with three parameters: (1) TC Block, (2) message key or list reference or list reference pointer in the range -7 < pointer < 0 , (3) varying length data area to hold the message text read from the file. On entry, pointers to these parameters are placed in registers and the type of request is determined (either key or pointer).

If it is a pointer, the key is obtained from EXPLIST (a push-down stack of keys). If the user wants just the key, control is returned to the caller. Otherwise, and if the second parameter is a key, the message file is searched for the Key.

If the key is not found, error flags are

set and control returned to the caller. Else, the text of the message is read into the user's area and the message key is reset to point to the next record of the file, if any, and control is returned to the caller.

5. TSPRMTC

This is the entry point called by any data base program to request a command from the user. On entry, an internal buffer is checked for the presence of a previously entered command. If one is there, it is returned to the caller as satisfying the prompt. If the buffer is empty, a message key passed as a calling parameter is used to access a message file to obtain the text of the message which describes the context of the prompt to the user. This message is displayed in the prompt area of the user's I/O device and the terminal is opened for input.

The response to this prompt must be a command. It may be the one requested by the calling program, in which case it is passed along. Or, alternatively, it may be any of the ''immediate'' commands which cause one of the immediate command processors to be invoked. After all activities associated with the immediate command are completed, the execution cycle beginning on entry to TSPRMTC is repeated until a satisfactory response is returned to the caller or until APOFF or END processing is initiated.

Consult the Command System User's Guide for details of command syntax.

6. TSPRMTD

This entry point is called by a data base program wishing to obtain user-entered data. On entry, the same internal buffer that holds commands is checked for a parameter string that may have been entered with a command. If data is there, it is parsed out of the string (in accordance with the syntactic rules outlined in the Command System

User's Guide) and returned to the calling program. If the buffer is empty or the next item in it is a command, a message key passed as a calling parameter is used to access a message file to obtain the text of the message which will explain to the user what data is requested. The message is displayed in the prompt area of the user's I/O device and the terminal is opened for input.

The response to this prompt may be data or any of the immediate commands. If it is data, it is parsed as above and returned to the user. The Terminal Control Block serves as a center for communicating information about the data between TSPRMTD and its caller.

If the response is an immediate command, this command and its associated parameters are treated separately from any user input intended for a data base program. When processing of the immediate command is complete, the cycle beginning on entry to TSPRMTD is repeated until a satisfactory response is received or until APOFF or END processing is initialed.

Consult the Command System User's Guide for the details of parameter syntax.

7. TSPRMTM

This entry point is called to display a message from the Message file on the user's terminal. No reply is asked for. Auxiliary subroutine entry points are called from various locations in TSUPER to perform prompting tasks.

The message filter MSGLEVEL in the user's profile determines whether or not informational (I-level) messages are displayed. Warning (W-level) messages are always transmitted.

The message ID filter MSGIDS specifies insertion of the message key between the message prefix and the text. MSGIDS=Y requests display of message keys.

MSGIDS=N implies no keys.

If the last output to the display area left residual data undisplayed, the ''MORE'' message is substituted for any command or data prompt message. The key of every message (except explanations) is placed in the EXPLIST area for reference by the EXPLAIN command.

8. TSPUT

TSPUT may be called one or more times by data base programs to format data (passed as a parameter) in a buffer for output. The data consists of a string of characters to be displayed on the user's terminal and an optional tag field which is appended to the beginning of the string. Formatting consists of manipulating the data so that it appears in a consistent and logical pattern on the screen.

On entry, TSPUT initializes pointers and work areas based on whether a restart, continuation or backwards put is indicated. After insuring there is sufficient room in the buffer to insert new data, a subroutine is called to move the tag and data string to the output buffer. This step is repeated until all data is in the buffer or the buffer is filled. An attempt is made to terminate lines between words and at punctuation.

On buffer overflow, if the caller does not want overflowed records inserted, all pointers are reset and control is returned to the caller. If partial records are inserted, control characters are appended, a TC Block variable is set to indicate the number of characters taken and the AUTOWRITE switch is checked. If it is on, control passes to the FLUSH routine, otherwise control is returned to the caller.

If all data is inserted with no overflow, the trailing position of the record is padded with blanks (to fill out a screen line) and control is

returned to the caller.

9. TSWRITE

This routine is called to flush the contents of the external storage named TSCREEN. After locating the area, control is passed to FLUSH, which outputs the data and returns control to the caller.

c. Subroutines

1. GETPR

This routine calls the PL/I controlled storage allocation routine ''IHESADD'' to obtain space into which the master PSECT may be copied. The caller's registers are saved in an area common to the copy routine so after the area is obtained a branch is taken to MOVECOPY and from there control is returned to the caller.

2. DELPR

This routine simply deallocates the external controlled storage allocated by GETPR. The PL/I service routine ''IHESAFF'' is called to perform this function.

On return, register 12 is set to point to the next area in the chain and control is returned to the caller.

3. GETSYN and GETDFAIT

These two subroutines primarily the same code, the differences being in the lengths of the parameter list used in the eventual call to an external program and the v-con which is posted in register 15 and points to the program which is called. GETSYN calls TSGSYN to obtain a synonym for a term. GETDFALT calls TSGDEF to obtain a default value for a parameter. On return from the respective calls, the length of the returned data is checked. If nothing came back, the data pointers are reset to point to the data used as a calling

parameter.

4. PULINEND

This subroutine is called by TSPUT to insert the proper end-control characters on each line of display output as it is moved into the output buffer. Screen lines are padded with blanks to fill out the line. Typewriter lines are terminated with an interpretive hex 15.

5. GIVITD and GIVITC

These two subroutines are called by the prompting routine to pass data to the user. If the prompt processing is in skip mode or the call was inadvertently done before an item-was found, the pass is not done. Otherwise SDPASS is called to move the data to the user's area.

On return, the passed data is excised from the input buffer. If it came from a parenthesized list, the list flag is set in the terminal Control Block and control is returned to the user.

6. SDPASS

SDPASS compares the length of the data or command passed from the input string with the receive area. If the item will fit the area, it is moved, otherwise a syntax error is noted and error processing is begun.

7. SDSYNCHK

This routine is called to check for certain syntax errors. If an error is detected, control is transferred to SYNNER to initiate an error control sequence. Otherwise, control is returned to the point of call.

8. RESETBUF

Preparing a buffer for input and initializing all flags associated with input parsing is performed here.

9. SDSTRIP and STRIP

SDSTRIP is called to delete leading and trailing quotes and leading and trailing blanks from an item passed as input to a calling program. If only blanks are to be deleted, entry is at STRIP.

10. PRKEYSAV

Inserts the key of a prompting message into a push-down list of message keys for reference by the EXPLAIN command.

11. IMCHECK

Whenever the user enters an immediate command, it is discovered by this subroutine. Comparing the entered command against a table of valid immediate commands, a ''hit'' leads to either signalling ''END'' or calling an external program to initiate processing. On return, the prompt routine is informed of the occurrences and the prompting cycle begins again.

12. SIGNAL and SIGNALC

Entry at SIGNAI causes preparations to call the PL/I service routine IHEERRD. Control then falls through to SIGNALC, which calls a pre-indicated routine and, on return, itself returns.

13. SETLDBA

This routine opens and initializes the DCB for the prompt message Library: NASIS.MESSAGES. Also, it issues a SETL to find a particular message key in the file. If the Key is not found in NASIS.MESSAGES a substitute message is written which indicates the message was not found.

14. GETMLF

GETMLF is the sister routine of SETLDBA. Its function is to read the text of a message record pointed to by a message key. Each record read is checked for the presence of a minus sign (-) or plus

sign (+) as its last character.

If there is a minus sign, the next record is read and appended to the first. If the last is a plus sign, the truncation bit in the TC Block is set to one(1) and control is returned to the caller.

15. MOVE

All extended data relocations are performed by this routine. In addition it is also used to blank-fill a data area and copy from one area to another.

16. PROMPT

On entry, if the user is in RESTART or RERUN mode the next record of input is obtained from the strategy dataset named in the external control block USERTAB. Otherwise pointers and constants are set in the I/O control block and MTT is called to do an I/D.

On return from MTT, the return code in register 15 is checked. If there was an error, attention interrupt or continuation the I/O is retried. Otherwise, the data is moved to a work area and control is returned to the caller.

17. EXIT

Returning to any program calling an TSUPER entry point is accomplished by passing through this code. The caller's TC Block is updated by moving our copy of it back into the caller's area. The PRV is restored in register 12 and control is returned by calling the PL/I service routine IHESAFA which releases our Dynamic Storage Area (DSA) and restores the callers registers.

F. CODING SPECIFICATIONS

1. Source Language

OS/360 Assembler Language.

2. Suggestions and Techniques

Not Applicable

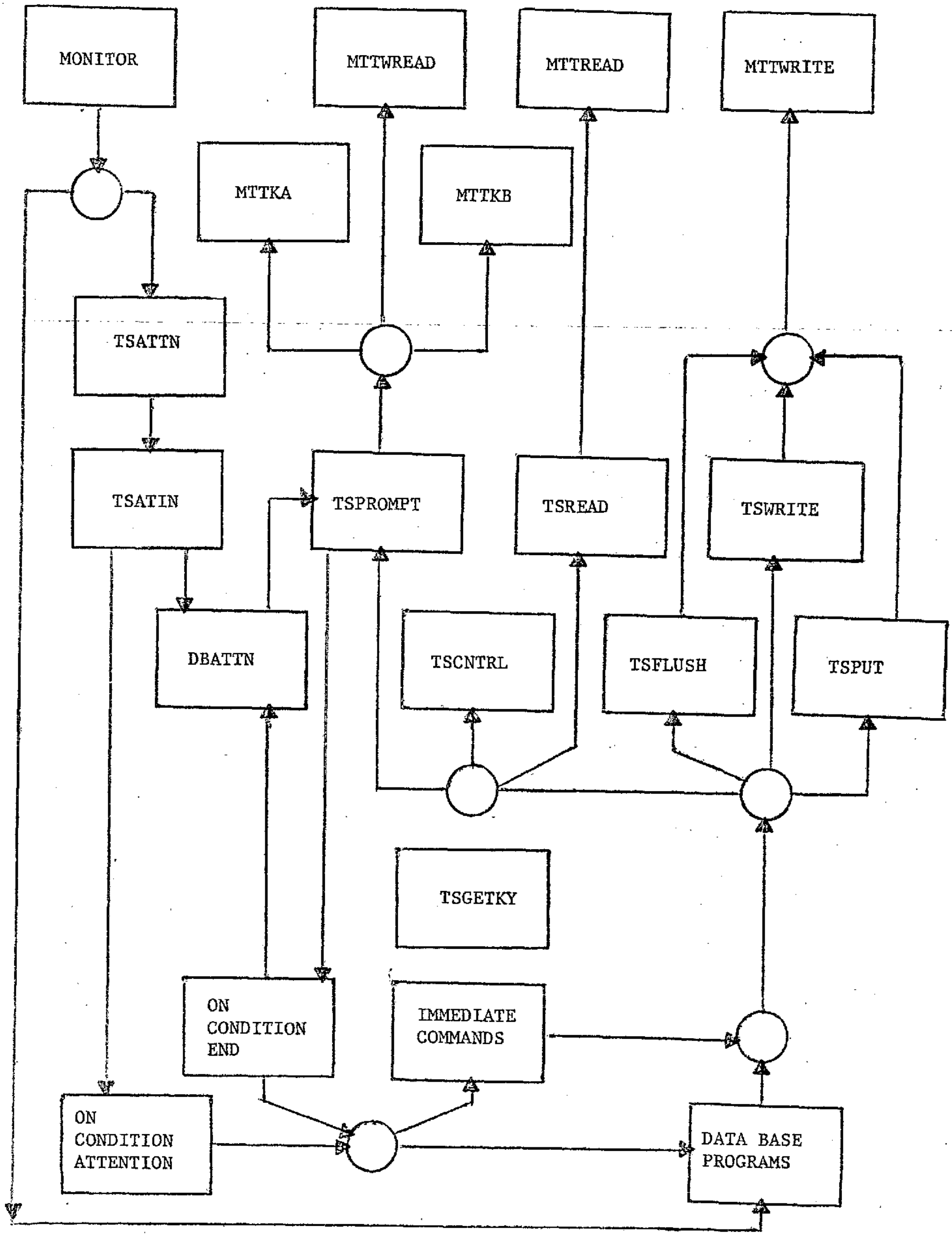


Figure 1. Terminal Support Organization Chart

C5

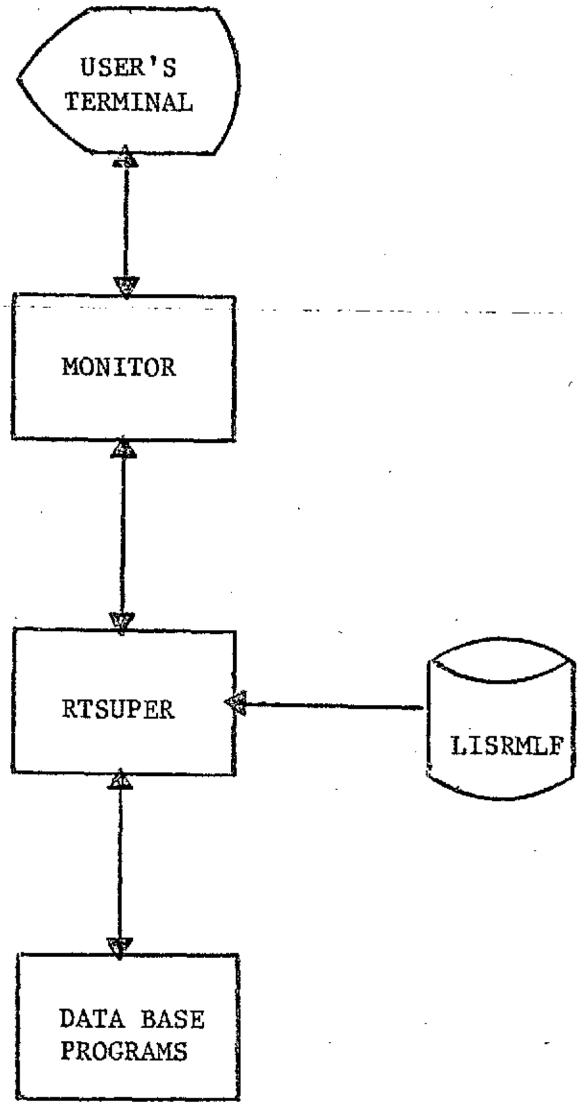


Figure 2. I/O Block Diagram

TOPIC E.3 - PLI-ASSEMBLER LINKAGE MODULE

A. MODULE NAME

Program-ID - NDBPLINK
Module-ID - DBPLINK

B. ANALYST

T. C. Moser
Neoterics, Inc.

C. MODULE FUNCTION

This module completes the linkage between a PL/I program and an assembler subroutine. It does so in such a way that the assembler routine may in turn call a PL/I subroutine and yet maintain the continuity of control necessary for proper PL/I linkage and communication. Another aspect of this linkage method is that it makes the module reentrant and recursive.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

Upon entry, the program initializes the variables it needs from the parameter list passed by the calling module. This data is used to obtain from PL/I library routine IHESADA a dynamic storage area (DSA) large enough to contain the register save area and a copy of the calling routine's pseudo PSECT.

Once this has been done, the program copies the calling programs pseudo PSECT to the DSA,chains the DSA into the pseudo register vector (PRV) and posts the DSA address in register 13. The program then initializes all of the base registers required.

Before exiting the program restores the remaining registers from the calling programs caller's savearea. It then chains the DSA into the

savearea chain and returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the OS Assembler language.

2. Suggestions and Techniques

Extreme care must be taken to ensure the fact that this program is completely reentrant and recursive. All operations should be performed in registers, or in the DSA obtained from PL/I.

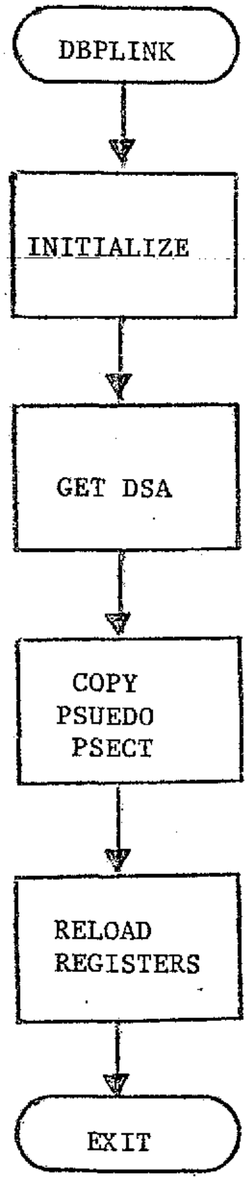


Figure 1. Top Level Flowchart - DBPLINK

TOPIC E.4 - ASYNCHRONOUS INTERRUPT PROCESSOR

A. MODULE NAME

Terminal Support - Attention Interface
Program-ID - NTSATTN
Module-ID - TSATTN
Entry Point - TSMATTN

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

TSATTN is the interface between the monitor and the terminal support supervisor TSUPER. Its function is to link the monitor to the TSUPER attention routine TSATIN. TSATTN is only called after an asynchronous interrupt resulting from the user depressing the attention key at his terminal.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

On entry, TSATTN performs OS standard linkage except that the address it picks up as its PSECT register points to a table of v-cons which are (in order): TSATIN and MTTUTAB. TSATIN is the entry point to Terminal Support's attention processing routine. MTTUTAB is a table which holds the user's pseudo-register vector (PRV).

After linking, TSATTN checks the interrupted register 13 to determine if it points to a PL/I

Dynamic Storage Area (DSA). If not, no further attempt is made to process the attention. That is, TSATTN returns to the monitor, effectively ignoring the interrupt.

When a valid DSA is found, the PRV is checked and if it is OK. the DSA registers are saved in an area provided by the monitor. TSATTN next calls TSATIN using the interrupted DSA as a savearea.

On return from TSATIN, the DSA regs are restored, the caller's registers are restored and control is returned to the monitor.

F. CODING SPECIFICATIONS

1. Source Language

OS/360 Assembler Language.

2. Suggestions and Techniques

The NASIS assembler macro library must be used to reference the User Information Table (TSUTAB). Also, entry linkage is standard OS/360 while calling linkage is standard PL/I.

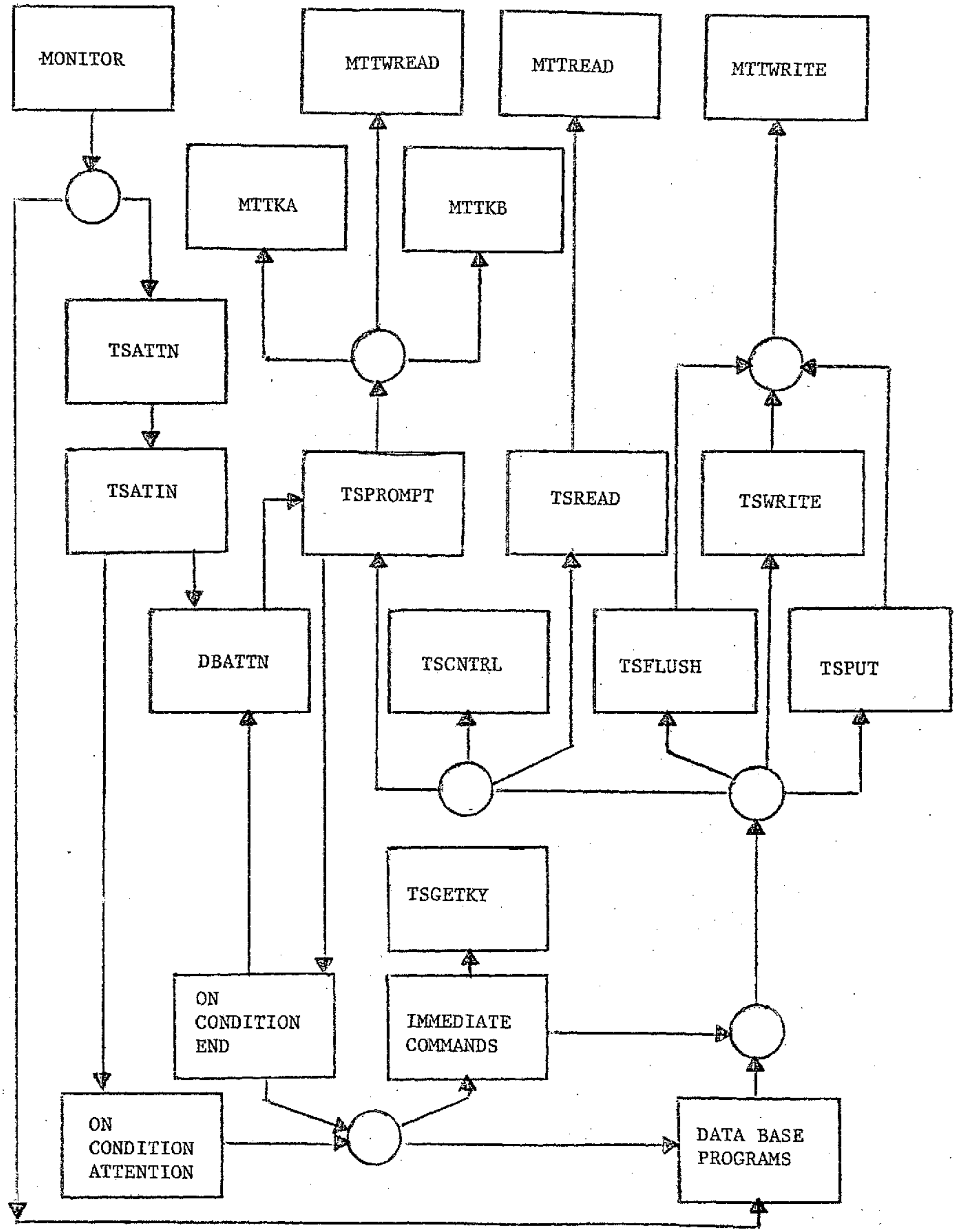


Figure 1. Terminal Support Organization Chart

TOPIC E.5 - ATTENTION PROMPTING PROGRAM

A. MODULE NAME

Terminal Support-Attention Prompting Program
Program-ID - NDBATTN
Module-ID - DBATTN

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

DBATTN is called by TSUPER to issue the command prompt ''-ATTN:'' and check the user's response thereto.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

Not Applicable

3. Output Data Sets

Not Applicable

4. Reference Tables

a. External tables

LISBMAC(USERTAB)

b. Internal Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

On entry, DBATTN checks the DISABLED switch in USERTAB. If attentions have been disabled, TSPRMTM is called to inform the user at the terminal and execution returns to the caller. If attentions are enabled, DBATTN sends a blank character out to insure that the carriage is in its home position, then issues a command prompt with the message ''-ATTN:'' to allow asynchronous commands to be entered by the user.

TSUPER intercepts all ''immediate'' commands except GO and calls the appropriate routine. If the user enters GO, null or any non-immediate command, DBATTN takes the following action:

a. GO or null - returns control to the caller, thus signifying the end of the prompting sequence.

b. Non-immediate command - ignores the user's response and reprompts as above.

If the END condition is raised while executing this module, execution control is returned to the caller.

F. CODING SPECIFICATIONS

1. Source Language

OS/360 PL/I

2. Suggestions and Techniques

Not Applicable

TOPIC F.1 - RETRIEVAL INITIALIZATION

A. MODULE NAME:

Program-ID - NDBINIT
Module-ID - DBINIT

B. ANALYST

John A. Lozan, William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

This module performs the initialization functions for the retrieval system and is the command director (prompting module) for retrieval.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

The program initially prompts for the FILE, NAME and ADDRESS parameters, and later, prompts for the retrieval commands.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program issues various diagnostic messages, where appropriate.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program references and optionally initializes the following tables,

USERTAB
FLDTAB
COLFORM
SEQFORM
SRCHTAB
VERBTAB
RETDATA
SETAB

E. PROCESSING REQUIREMENTS

1. TOP LEVEL FLOWCHART

See Figure 2

2. Narrative

Upon entry the program initializes itself and the terminal support facilities. It calls DBJOIN to process the file parameter and prompts for the NAME and ADDRESS parameters. The parameters are all verified and saved for later reference.

The program then initializes the retrieval data table, RETDATA. The set table, SETAB is then initialized, the dataplex is opened for input and the field table, FLDTAB, is initialized.

The program then initializes its verb table, including the addition of any user defined commands. Now the program prompts the user for a retrieval command. If the command entered is not valid, a diagnostic message is written to the user and he is reprompted.

If the command entered was not END or RETRIEVE, the program calls the entry point specified for

that command. If the called module requests another module, the other module is called. If necessary the original module is recalled. Then the user is prompted for his next command. If the user entered END or RETRIEVE, the retrieval session is terminated by closing the data base, erasing the sets and the formats. All searches are cancelled. If the user entered RETRIEVE, the program branches back to initialize itself for a new retrieval session. Otherwise, the program is terminated.

Due to the complex relationship of the two modules performing the DISPLAY function (DBDSPL and DBDSPLA), the DISPLAY paging entry point, DBDSPLP, is within DBINIT. Only the program calling code is utilized and a return is made following the completion of the DISPLAY modules.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the IBM PL/I language.

2. Suggestions and Techniques

Not Applicable

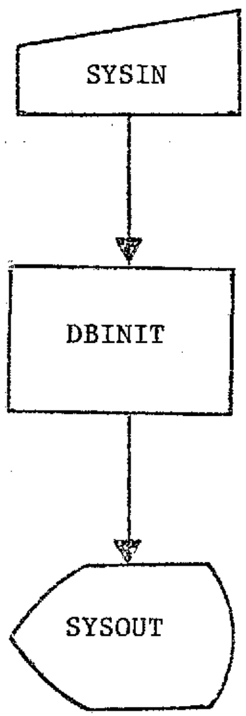


Figure 1. I/O Block Diagram

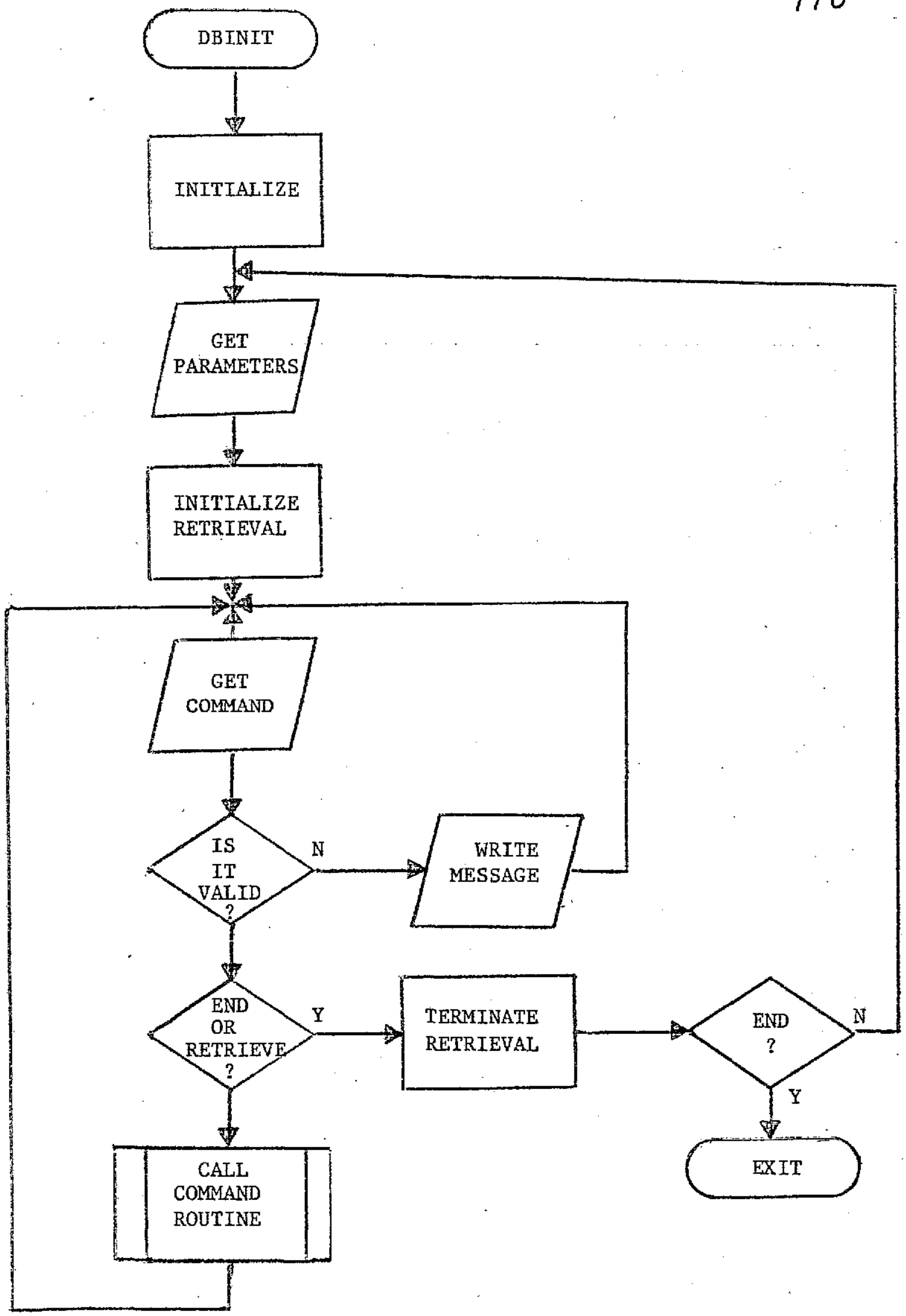


Figure 2. Top Level Flowchart

TOPIC F.2 - RETRIEVAL FIELDS COMMAND

A. MODULE NAME

Program-ID - NDBFLDS
Module-ID - DBFLDS

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module displays a formatted listing of the field names of the file currently being accessed by the user.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

The routine prompts for the parameter associated with a PAGE command.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces a formatted list of

field names.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

FLDTAB-The program extracts some of its information from FLDTAB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBFLDS

At this entry point the parameter is checked for a paging request; if paging, the processing continues with the label DBFLDSP (see below). Otherwise, the program initializes the screen and paging status data. It extracts the database name from FLDTAB. The program then, repetitively, retrieves field names with calls to the field utilities in DBFLDU - FLDGET, FLDCLAS, FLDCTRL, and FLDSKEY. It flags each field that has an inverted index. It posts the field names tc the screen. When the list of names has been exhausted, or the screen has been filled, the screen is displayed to the user, the paging status data is posted and the program is terminated.

b. DBFLDSP

At this label the program is re-initialized using the paging statts data. If more data remains, the program tranches to the proper routine to build the next screen image. Otherwise, a diagnostic message is written to the user and the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

   The module is written using the IBM PL/I Language.

2. Suggestions and Techniques

   Not Applicable

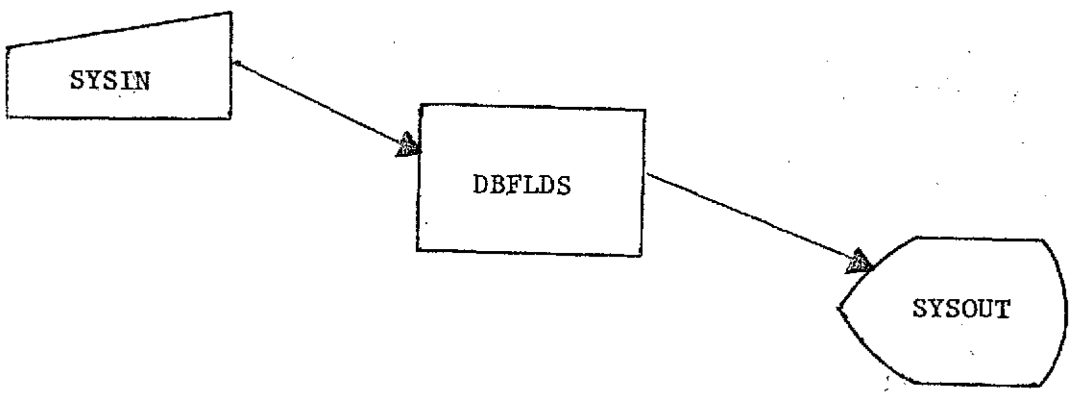


Figure 1. I/O Block Diagram

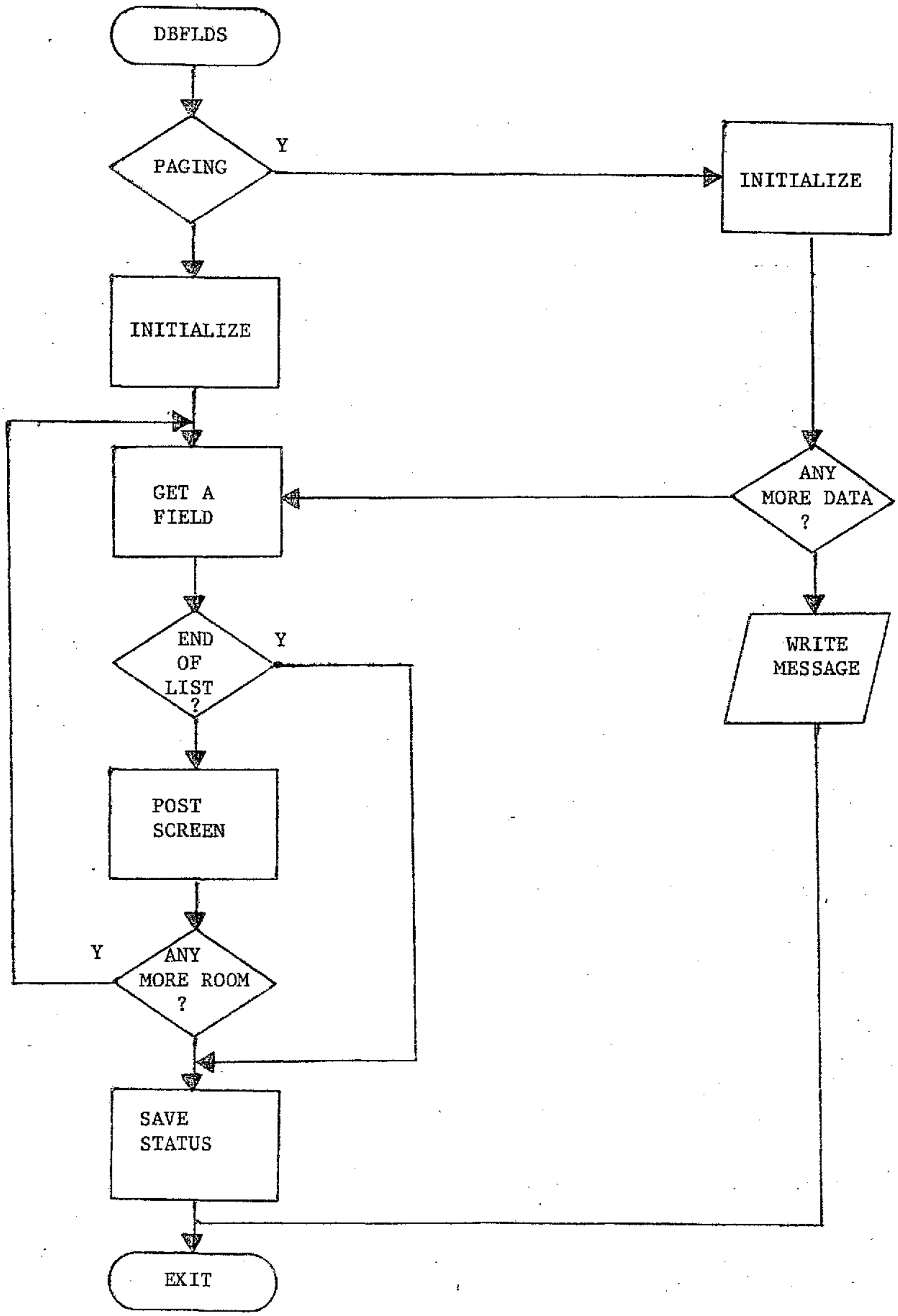


Figure 2. Top Level Flowchart - DBFLDS

TOPIC F.3 - RETRIEVAL EXPAND COMMAND

A. MODULE NAME

Program-ID - NDBXPND
Module-ID - DBXPND

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module displays to the retrieval user, a formatted listing of a cross section of an inverted index at and beyond a specified term.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The inverted index files of a dataplex are used as a source of data by the program.

d. On-Line Terminal Entries

The program prompts for the TERM and INDEX parameters.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces a formatted listing of the index records read.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program uses the following tables as a source of data and as a means of data control,

USERTAB
FLDTAB
EXPTAB
EXPTERM

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBXPND

At this entry point the parameter is checked for a paging request; if paging, the processing continues at label DBXPNDP (see below). Otherwise, the program initializes itself to perform a new expansion of an index file. The program initializes the screen and the data storage table EXPTAB.

The program then prompts the user for the TERM and INDEX parameters. The parameters are validated and the program gets ready to read the index (or anchor) file specified. The first read of the file is for positioning, based upon the term entered by the user. If more data remains on the file, the program begins reading records, saving the data in EXPTAB and posting them on the screen. The relative E-number is computed and also posted. If an end-of-file is encountered, an indication is posted on the screen. At this point, or when the screen is filled, it is displayed to the user, the

paging status data is posted and the program terminates.

If any errors are encountered, a diagnostic message is written to the user and the program is terminated.

b. DBXPNDP

At this label the program re-initializes itself using the paging status data. If more data remains to be displayed the program branches to the appropriate point to begin reading the index and building the new screen image. If no more data remains, a diagnostic message is written to the user and the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the IBM PL/I language.

2. Suggestions and Techniques

The AREA facilities of PL/I should be used to organize the term data stored in EXPTAB to optimize file access and data storage.

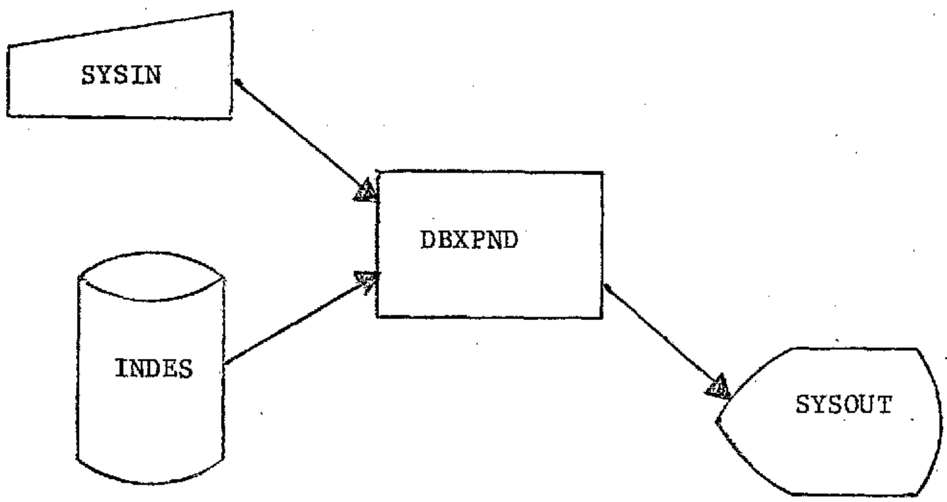
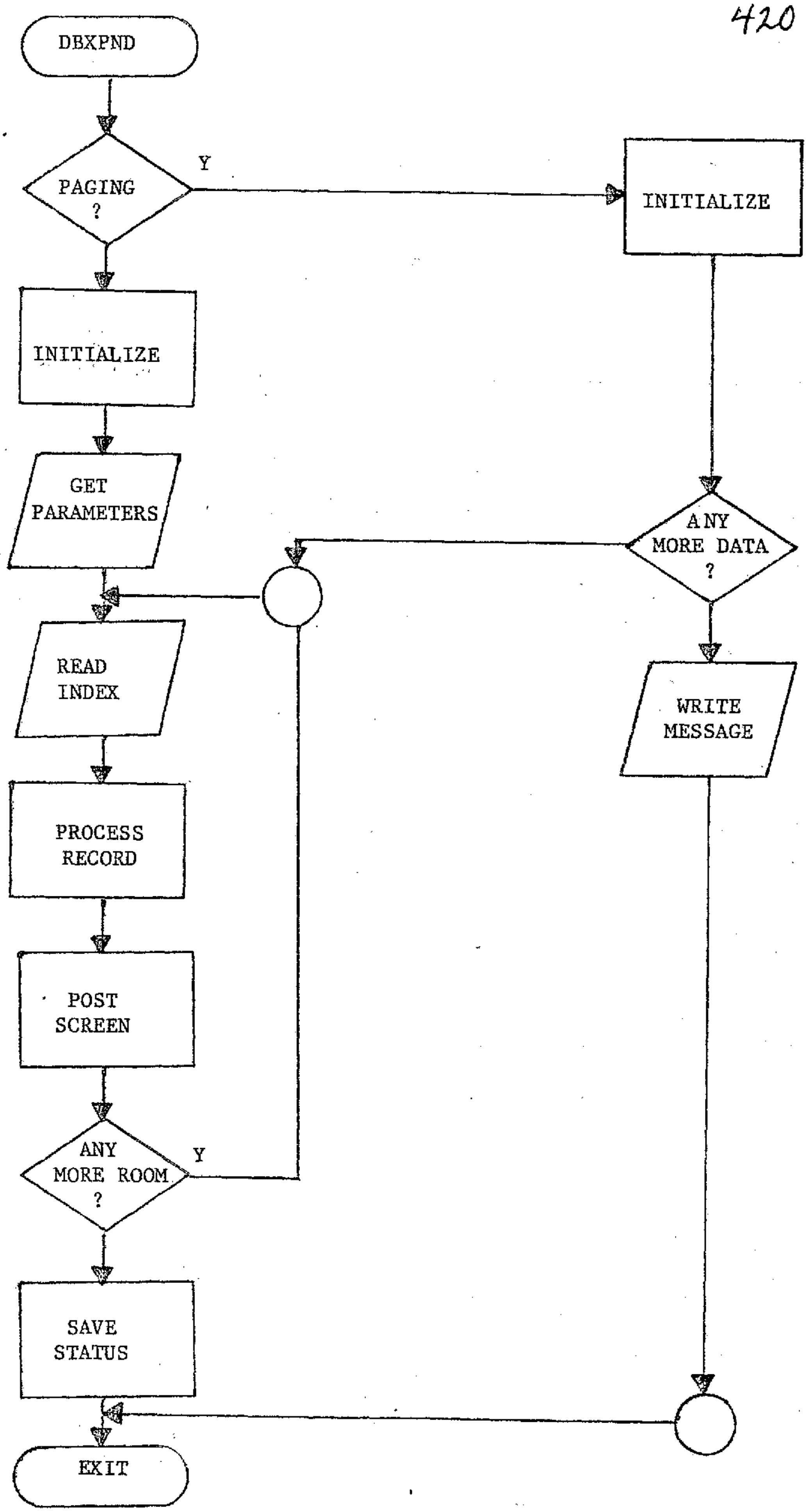


Figure 1. I/O Block Diagram


DBXPND
PAGING ?
Y
INITIALIZE
INITIALIZE
GET PARAMETERS
ANY MORE DATA ?
READ INDEX
WRITE MESSAGE
PROCESS RECORD
POST SCREEN
ANY MORE ROOM ?
Y
SAVE STATUS
EXIT

TOPIC F.4 - RETRIEVAL, Select Command

I. SELECT

A. MODULE NAME

Retrieval, SELECT Command
Program - ID - NDBSLCT
Module - ID - DBSLCT
Entry Points (DBSLCT0,DBSLCT1,DBSLCT2)

B. ANALYST

O. Kirt Hearne
Neoterics, Inc.

C. MODULE FUNCTION

The SELECT command format is:

SELECT expression,field,replace,method

The SELECT command outputs the expression and the number of citations (record keys) for which the expression applies. A set number or S-number is assigned to the expression, and the command string is entered into the next available line in the current search strategy.

The expression parameter (keyword=EXPR) is a boolean combination of terms which define a set. If all fields referenced are indexed, the expression is evaluated immediately and a set-number assigned. If a field in the expression is not indexed or a previous S-number is referenced, a search entry is constructed and saved, and an S-number assigned.

Only a single non-indexed field is allowed in a single SELECT expression.

The field parameter (keyword=FIELD) is used by SELECT to resolve any values in the expression which are not directly related to a fieldname within the expression.

The replace parameter (keyword=REPLACE) is a previously defined S-number which is to have its expression replaced by the current expression.

The method parameter (keyword=METHOD) is used to force a search on indexed fields. To do this, "SEARCH" must be entered as the method parameter.

Note that only a single field may be referenced in this case.

SELECT will prompt the user if the expression is missing, or the field parameter is missing and found to be needed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The descriptor files and the index files may be referenced by the SELECT command. The descriptor file is used to obtain the data set name of the subject term index file. The index files are used to obtain a list of accession numbers associated with a particular subject term.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The command string, as it is entered, is saved in the region containing the current strategy using the routine PSTRAT.

b. On-line Terminal Displays.

The following is displayed if a set is successfully produced from the expression:

(1). A unique set number or S-number.

(2.) The number of citations (or keys) in the set

(3.) The expression, with:

(a.) E-numbers replaced with the corresponding "fieldname=value".

(b.) Values which return a null are notated with special symbols, as: AGE =>>'9999'<<.

(c.) If the resultant set consists of subfile keys, the expression will be displayed with the subfile name, as:

(FROM:subfilename) expression

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. EXPTAB

b. FLDTAB

c. MFCB

d. PARSED

e. SETAB

f. SRCHTAB

g. TC

h. USERTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. NARRATIVE

The SELECT command outputs the expression and the number of citations (record keys) associated with that expression. A unique set-number or S-number is assigned.

The input expression is a boolean expression made up of set-numbers, S-numbers, values, E-numbers or range forms of these terms.

The SELECT command is processed in three phases:

1. Parsing
2. Expression analysis
3. Execution of SELECT "instructions"

SELECT parses the expression in three passes. The first pass recognizes and marks as such, letter strings, digit strings, operators, special characters and delimiters. Quoted strings are recopied to remove any double quotes.

The second pass recognises primary elements such as S-numbers, E-numbers, set-numbers, values, and field names. Field names are marked as indexed or non-indexed.

The third pass recognises groupings of elements such as range forms and associates each value in the expression with the proper field name. If necessary a prompt with the keyword "FIELD" is done to obtain the field name. This pass also sets up SELECT execute phase instructions for the creation of sets from basic terms such as a set-number.

Also, during the third pass, a non-indexed field name appears in the expression, the proper entries are made in SRCHTAB to provide for the search to be executed later.

All information found during the first three passes is entered into PARS_TAB and PTAB_INFO. The original expression, recopied quoted strings, and other necessary character strings are all contained in WAS. Each element in PARS_TAB contains an index (IDX) into was to note the position of the item

described.

The next phase of SELECT analyses the expression algebraically and builds execute phase instructions to perform the proper operations. If a search is required instructions are built to post final entries in SRCHTAB, before the search, and to retrieve information from SRCHTAB, after the search, for final evaluation of the expression.

During expression analysis, the ANDing of a search term with another set is noted, and instructions are created to cause the search to occur only within the set ANDed with the search term.

After the second phase all is ready for final evaluation of the expression by execution of the previously created instructions. At this time the input command, with parameters, is reconstructed and posted in the CURRENT_STRATEGY data set.

If a search is required, all SELECT tables and instructions are stored for use at the time of search execution. An S-number is assigned and this number, with the expression, is output to the terminal.

If no search is required, the execution phases of SELECT is invoked. The instructions built earlier are now executed. Sets are created, combined, and a altered as the expression dictated, until the final resultant set is obtained. This set is assigned a unique number and posted into SETAB through the use of the DBPSET routine which also sends a line describing the set (set- number, size, expression) to the terminal.

When the user enters the EXECUTE command to invoke the search, the DBEXSR program is given control. This routine contains all of the actual search logic, however repetitive calls to SELECT (DBSLCT2 entry point) are made. The execute phase instructions are used by SELECT to control the search.

During a search each previously defined S-number has associated with it an

instruction list. The first instruction in the list for each S-number is a "branch" initialized to point to the second instruction in the list. When SELECT is first given control, each instruction list is executed until an S-number or a search term instruction is encountered. The search instruction posts proper final information to SRCHTAB and in both cases execution of the instruction list is suspended. A new branch point indicating where to resume execution is stored in the "branch" instruction at the top of the list.

When all instruction lists have been executed as far as possible, control is returned to DBEXSR for the actual search to take place. After this SELECT is called again and instruction execution is restarted. Some S-numbers and searches may now be evaluated. Again each instruction list is executed until an undefined S-number or search term is encountered or an actual set is created and posted. Again control returns to DBEXSR. This process continues until all instruction lists terminate by posting a set.

The SEARCH is implemented simply as an additional entry (DBSLCT1) into SELECT. The command format is the same as that for the SELECT command, thus a valid SELECT expression may be used.

DBSLCT1 is the entry point for the SEARCH. This command first gets and verifies the set number or S-number on which a linear search is to be performed. SEARCH then prompts the user for the rest of the search expression to be performed to the specified set. Once the search expression is entered, then SEARCH passes this information to the search option part of the SELECT command. When control is returned to SEARCH, it then prompts the user for another search to be performed on the same set as before. This loop continues until the user enters a null response to the search expression prompt, at which time control is passed to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

The SELECT command module is written in the IBM/360 PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base and the TSPL/I language extension is used to handle all communication with the terminal.

2. Suggestions and Techniques

Not Applicable

II. SELECT, THE SEARCH OPTION

A. MODULE NAME

Retrieval, SELECT Search Option
Program - ID - NDBSLCT
Module - ID - DBSLCT

B. ANALYST

O. Kirt Hearne
Neoterics, Inc.

C. Module Function

The SELECT search option is a feature of the SELECT command which guides the user through a search strategy. The SEARCH command is used to define a set or pseudo-set to be used as the search universe.

The user is then prompted for linear search expressions with the phrase:

SELECT (Set-number S-number) IF:

The reply is of the same format as the SELECT command itself:

expression,field,replace,method

where the parameters have the same meaning as with the SELECT Command.

The set-number or S-number defined by the SEARCH command is added along with an AND boolean operator to the left end of the expression entered in response to the SELECT IF prompt. The resultant expression is then sent directly to the SELECT command processor.

1. Reference Tables

   a. EXPTAB

   b. FLDTAB

   c. MFCB

   d. PARSED

e. SETAB

f. SRCHTAB

g. TC

h. USERTAB

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

If a terminal is the source of search parameters as previously defined, the TS system will apply default values, if available, to the parameters when no values are entered.

3. Output Data Sets

a. Output Files

Using the PSTRAT routine, the command string, as it is entered and validated, will be saved in the region CURRENT_STRATEGY.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The SELECT Search command format is:

SEARCH expression,field,replace,method

which results in a set-number or S-number. The user is then prompted for a linear search:

SELECT (Set-number S-Number) IF:
expression
field,replace,method

The set-number or S-number is added, along with an AND operator to the expression and the result is sent to the SELECT command processor. Thereafter all processing is the same as for any SELECT expression.

After the expression is processed, the user is again prompted with the SELECT IF prompt. This continues until a null is entered.

F. CODING SPECIFICATIONS

1. Source Language

The SELECT Search command is written in the IBM/360 PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base, and the TSPL/I language extension is used to handle all communications with the terminal.

2. Suggestions and Techniques

Not Applicable

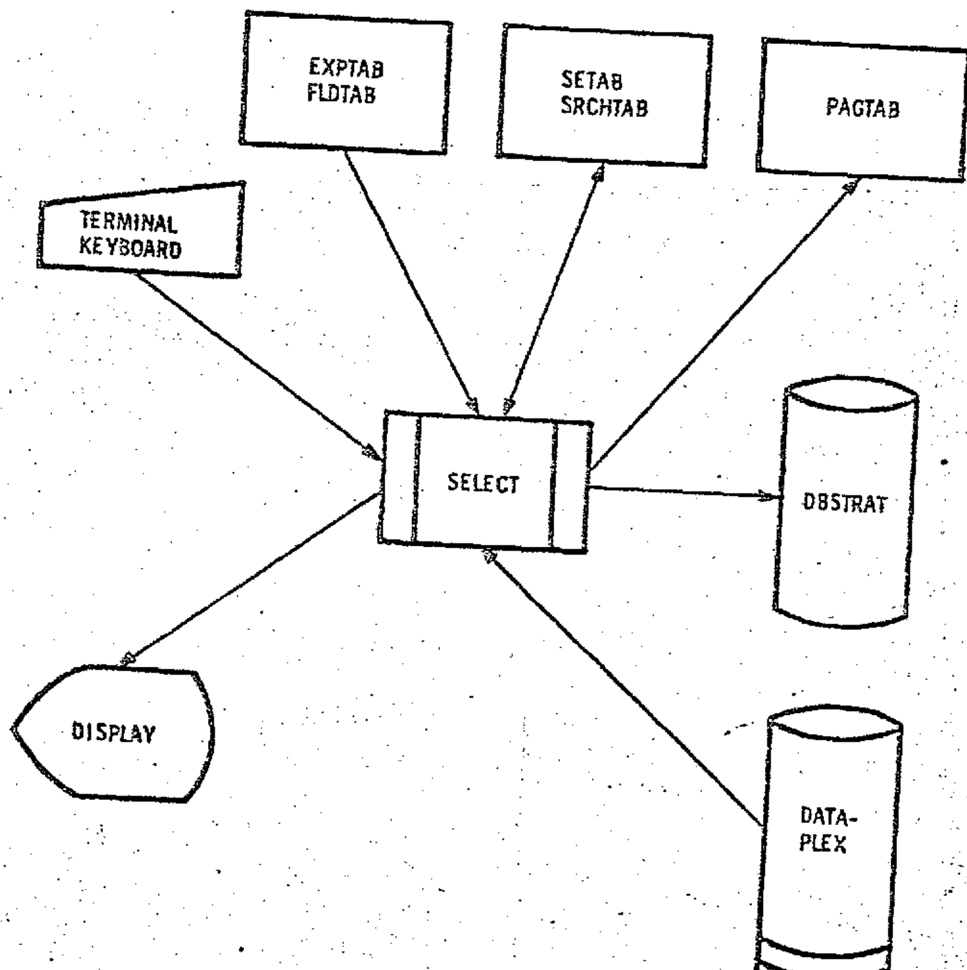


Figure 1. Block diagram.

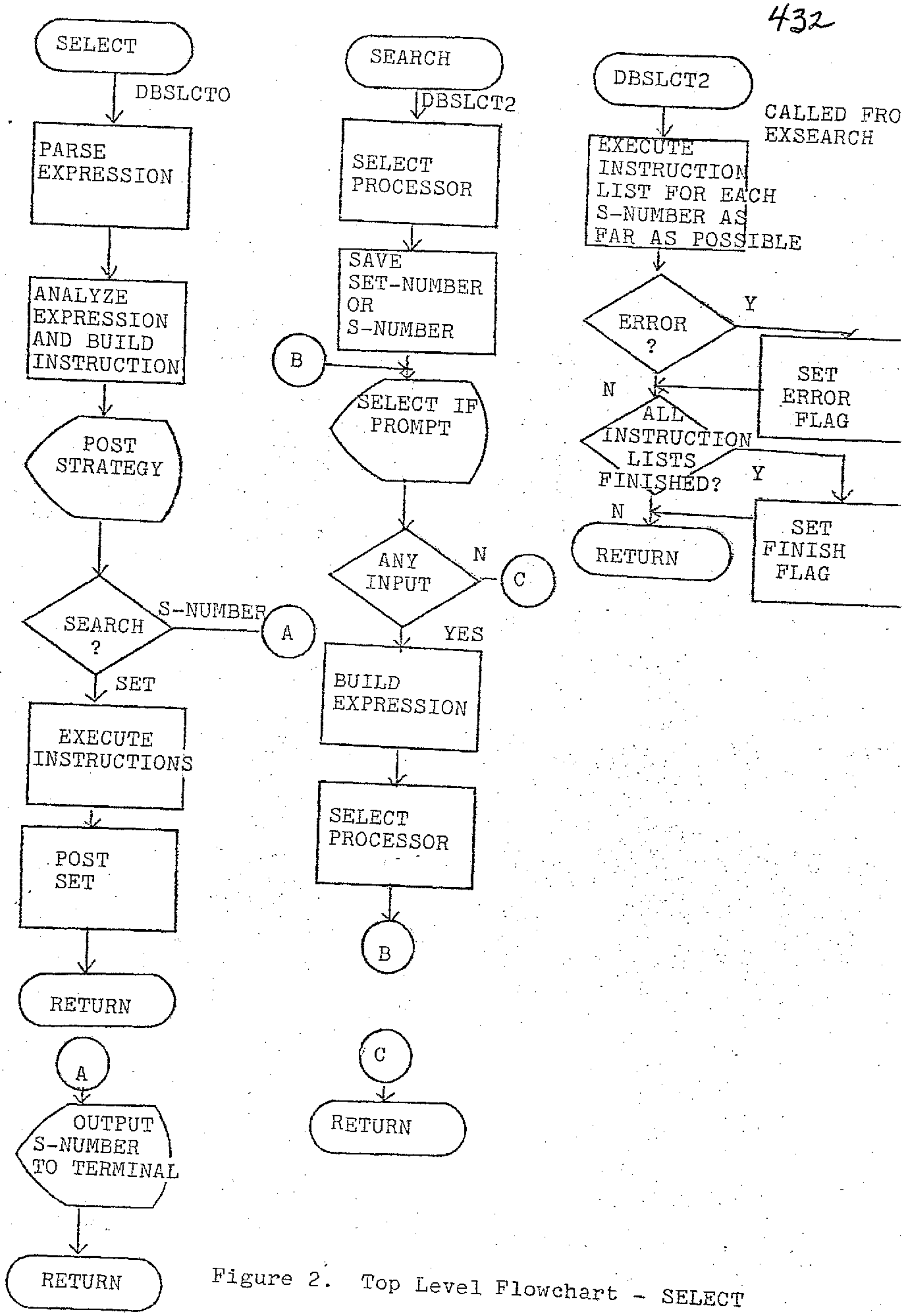


Figure 2. Top Level Flowchart - SELECT

TOPIC F.5 RETRIEVAL DISPLAY COMMAND

A. MODULE NAME

Retrieval, DISPLAY Command (module 1 of 2)
Program-ID - NDBDSPL
Module-ID - DBDSPL

B. ANALYSTS

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The DISPLAY command is a routine whose purpose is to allow the retrieval system user to have designated data for a given set to be displayed on a terminal. Like the PRINT command, the user may specify the format of the output as the citation number, the citation, the abstract, or the full text for any item contained in a set which has been previously selected. Optionally, the user may prespecify a format of his own, using the FORMAT command, to govern the DISPLAY command. One set-number is reserved for special purposes in the system. Set-number 0 is a logical reference to the entire anchor file. The PAGE command also calls the DISPLAY command in order to create additional displays, logically, before and beyond the current one. The calling sequence is: DISPLAY set-number, format, item, type or, alternately, DISPLAY citation#, format.

This module performs preliminary analysis of DISPLAY parameters and file positioning; it then calls the second DISPLAY module DBDSPLA via DBINIT. Refer to the DBDSPLA Program Design Specifications.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

c. Input Files

The anchor and associated files of a dataplex will be input to the DISPLAY command. The complete description of the files in a data base is found in the Data Set Specification Section of the Worbook.

d. On-line Terminal Entries

A terminal is the most likely source of the parameters which are passed to the DISPLAY command. The parameters available to the DISPLAY command are set or citation number, format, items, and type. The NASIS system will apply default values to the parameters, if they are available, when no original values are entered.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

The DISPLAY command will output a partially-formatted display of the items in a set or for a specific citation number. For sequential formats, each field is started on a new line, and the key field is always on the first line below the header information for a particular display. For columnar formats, the fields from each record are arranged across one or more lines in columns. The content of the display depends upon the format code entered as the second parameter.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COLFORM

The DISPLAY command refers to a COLFORM table

when a columnar format is referenced.

b. USERTAB

This table contains user-oriented and status information.

c. FLDTAB

The DISPLAY command refers to FLDTAB to locate the appropiate sequential (SEQFORM) or columnar (COLFORM) format table.

d. RETDATA

This table contains data fields unique to the retrieval sub-system.

e. PLEX

The DISPLAY command uses a DBPL/I file called PLEX for all of its retrievals from the dataplex.

f. SEQFORM

The DISPLAY command refers to a SEQFORM table when a sequential format is referenced.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Display

The DISPLAY command is called by the NASIS system by the director.

b. Accept Parameters

Since the parameters are not passed to the DISPLAY command, by the director, they are retrieved via Terminal Support (TS). The first parameter is either a "set-number", or a "citation #". The second parameter is "format" code, the third is an "item" number and the fourth is the "type" code. The last three parameters are optional. The "set-number" is a one or two digit number and

is not likely a default value since it will change for every command. The "citation #" is a character string, which is not likely to have a default. If no entry is made and no default exists, then an error is reported and control passed back to the calling routine. The "format" code is a value of 1 to 25 designating a sequential format or F1 to F25 designating a columnar format, or a format name representing one of the above format values or a fieldname. If no entry or default is present, the value "2" is provided for anchor key sets or "5" for subfile sets. The "item" parameter designates the member of the specified set. The entry is a character string having a numeric value. If no entry or default is given for this parameter, the first item in the set is displayed. The "type" code indicates whether the user wants subfile information to be displayed continually following the anchor data, and if so, whether the data fields of each subfile record are to be exhausted sequentially or the data field values are to be exhausted across subfiles before proceeding to the next field. An invalid entry is reported before returning control to the calling routine. If all parameters have valid values, then execution continues with the next section.

The DISPLAY command is placed as the next record in the strategy data set by a call to the save strategy routine. The parameters to this subroutine are the word DISPLAY and its parameters in their normal order.

c. First Page Initialization

Depending on the "class" of the first parameter, certain specific initialization is necessary. If the parameter is a dataplex key (class 1), e.g., a citation number, then the anchor record is read and a heading prepared. If the parameter is a set number (class 2), the relative key is taken from the set and used to read the anchor record and a heading prepared. Control is transferred to Section (f) below.

d. Page DISPLAY

The DBDSPL module is entered with a parameter value indicating paging; control is passed to

the DBDSPLP label. (The paging entry point for DISPLAY is within DBINIT to utilize the code there to handle the multi-module transient interface.) The paging direction and mode are indicated by the PAGE parameters.

e. Validate Next Page

Depending on the "class" of the first parameter to the original DISPLAY command and the paging direction, certain specific validation and initialization is necessary. If the page requested has been seen before, it need not be regenerated, but may be retrieved from based storage, where it was saved, and control can be transferred to Section(g) below to display it. When non-contiguous skip paging is being done, the relative key is taken from the set and the anchor record read.

f. DBDSPLA

The remainder of the processing for the DISPLAY command is handled by the DBDSPLA module. This module is called via the DBINIT Transient Module Interface convention. Necessary information is retained in a common structure called DSPLCTL.

g. Return

Do a normal return to the calling routine.

3. Submodules Required

a. DB - data base package

b. PSTRAT - save strategy

c. TS - terminal support package

d. DBSETU - set information package

e. DBFLDU - field utilities

F. CODING SPECIFICATIONS

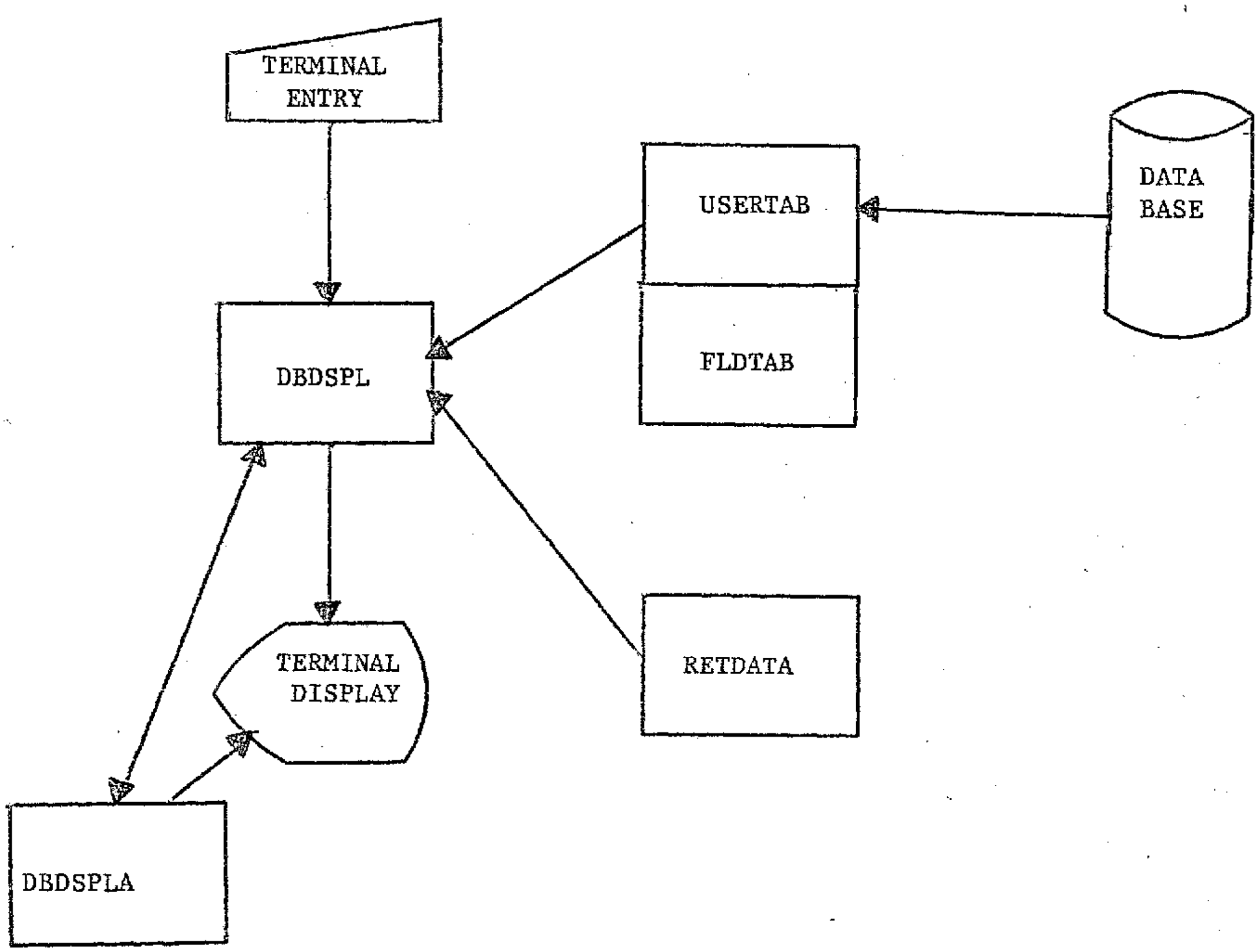
1. Source Language

The DISPLAY command is coded entirely with the IBM PL/I programming language. The DBPL/I language

extension is used to handle all access to the files in the data base. The TSPL/I language extension handles all instances of communication with the terminal.

2. Suggestions and Techniques

Not Applicable


TERMINAL
ENTRY
USERTAB
FLDTAB
DATA
BASE
DBDSPL
TERMINAL
DISPLAY
RETDATA
DBDSPLA

TOPIC F.6 - RETRIEVAL PRINT COMMAND

A. MODULE NAME

Retrieval, PRINT and CANCEL commands
Program-ID - NDBPRNT
Module-ID - DBPRNT
Entry Point (DBPRNT)

B. ANALYSTS

Frank E. Reed
William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

The PRINT command is a routine whose purpose is to allow the retrieval system user to have designated data for a given set listed on a high-speed printer. Like the DISPLAY command, the user may specify the format of the output as the citation number, the citation, the abstract, or the full text for any item or range of items contained in a set which has been previously selected. Set number 0 is a logical reference to the entire anchor file. Optionally, the user may prespecify a format of his own, using the FORMAT command, to govern the PRINT command. All of the uses of the PRINT command during a single terminal session will be accumulated and printed out as one continuous output for the user to pick up at a later time. All prints are queued for the user to be printed later by the DBA with the Print Monitor (DBPRINT). Prints issued on S-numbers (pseudo-sets) are not queued but merely deferred until after the EXECUTE command; parameters are saved in SPRNTAB. The command sequence is: PRINT set-number, format, item(s), type, copies, or, alternately, PRINT citation#, format.

The CANCEL command terminates an active SEARCH specification or deletes one or more queued prints called BSN's. The command sequence is: CANCEL range.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

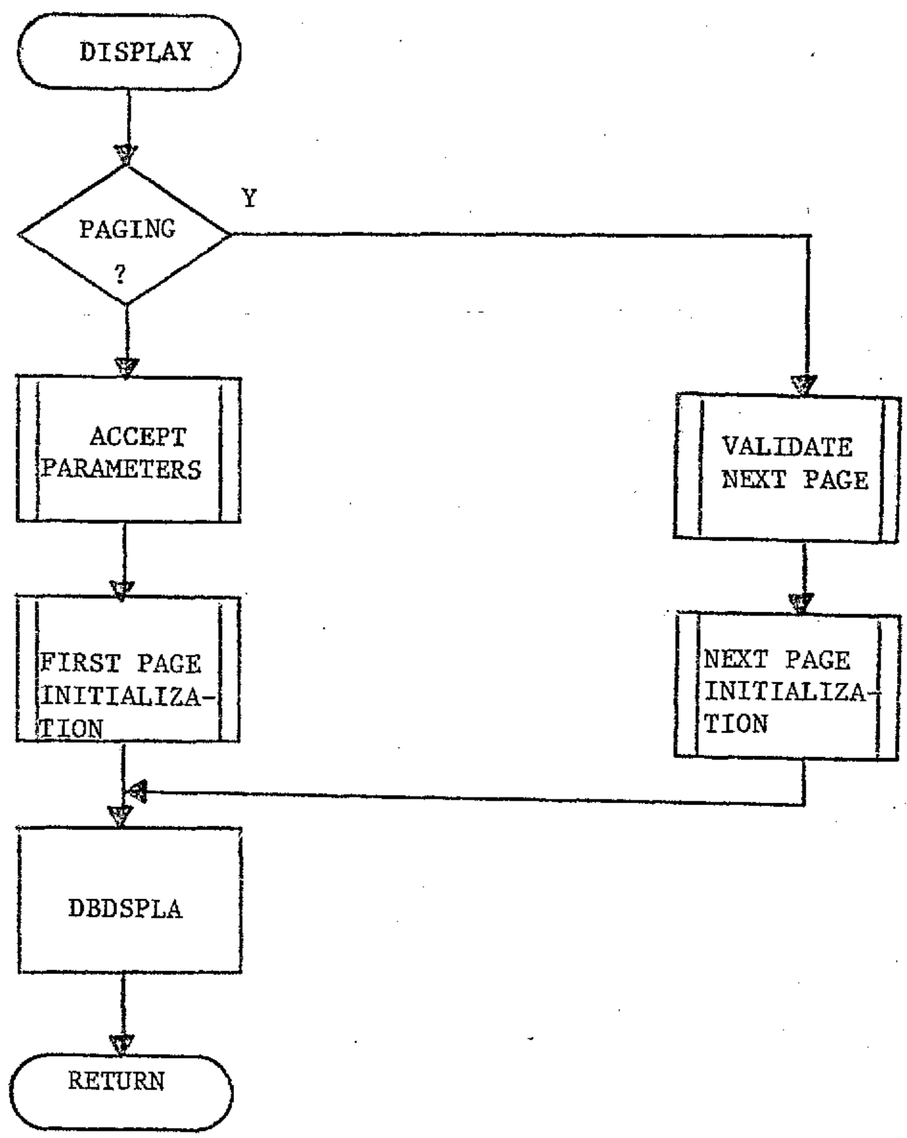


Figure 2. Top Level Flowchart

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The anchor and associated files of a data base are input to the PRINT command.

d. On-line Terminal Entries

The parameters available to the PRINT command are "set-number" (or "citation number" or "s-number"), "format", "items", "type", and "copies". NASIS will apply default values to the parameters if they are available, when no original values are entered.

3. Output Data Sets

a. Output Files

The output of the PRINT command consists of the strategy library containing the queued BSN.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COLFORM

The PRINT command refers to a COLFORM table when a columnar format is referenced.

b. FLDTAB

The PRINT command refers to FORMTAB to locate the appropiate sequential (SEQFORM) or

columnar (COLFORM) format table.

c. USERTAB

This table contains user-oriented and status information.

d. PLEX

The PRINT command uses a DBPL/I file called PLEX for all of its retrievals from the data base.

e. SRCHTAB and ENTRYDEF

These tables contain S-number information.

f. SEQFORM

The PRINT command refers to a SEQFORM table when a sequential format is referenced.

g. RETDATA

This table contains data fields unique to the retrieval system.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Print/Cancel

The PRINT command is called by the director; processing continues with Section (b). The CANCEL command is called by the director with a non-zero parameter value; processing continues with Section (d).

b. Accept Parameters

Since the parameters are passed to the PRINT command through Terminal Support, they are arranged in a keyword or predefined order. The first parameter is either a "set-number", as defined by a SELECT command, or a "citation #" or an "S-number" as defined by a SELECT-IF command. The second parameter is a "format" code, and the third is an "item"

number or range of numbers. The fourth parameter is a "type" code governing the form of sequential formatting, and the fifth parameter is a "copies" option for up to 9 copies. The latter two parameters are optional. The set number will be a one- or two-digit number and will not likely have a default value since it changes for every command. The "citation #" is a character string which also will not likely have a default. If no entry is made and no default exists, then the error is reported and control passed back to the calling routine. The "format" code is a value of 1 to 25 designating a sequential format or F1 to F25 designating a columnar format or a format name representing one of the above format numbers. If no entry or default is present, the value of two is provided for anchor key sets or four for subfile sets. The "item" parameter is not required when the "citation #" is entered as the first parameter; otherwise, it designates the member or range of members of the specified set. The entry is a character string of one to eleven positions. When a range of items is entered, the two values are separated by a hyphen. If no entry or default is given for this parameter, all of the items in the set are printed. The "type" parameter may be 1, 2, or 3; a type of 1 is the default. The "copies" parameter may have a value up to 9, with 1 as the default. An invalid entry will be reported before control is returned to the calling routine. If all parameters have a valid value, then execution continues with the next section.

The PRINT command is placed as the next record in the strategy data set by a call to the save strategy routine. The parameters to this subroutine are the word PRINT and its parameters in their normal order.

If the first parameter was an S-number, the provided parameters are entered in a SPRNTAB table for later use and processing continues with Section (e) below.

c. Queue the Print

The data pertinent to produce the print is then stored in the user's strategy file as a

batch sequence number (BSN) to be later processed by the Print Monitor (DBPRINT). Processing continues with Section (e).

d. Cancel Processing

The CANCEL parameter may have the value of a BSN (i.e. 1, 2, etc.), *ALL meaning all outstanding prints queued for the user, or the string 'SEARCH'. For a CANCEL SEARCH command the director is returned to with an indication made to call the DBEXSR module to perform the CANCEL. The BSN cancels are handled thusly. The BSN is verified to exist and then to be the user's. Then the strategy region containing the BSN(s) is deleted. Processing continues at Section (e).

e. Return

When all processing for the PRINT/CANCEL command has been completed, control is returned to the calling routine.

3. Subroutines Required

a. DB - data base package

b. PSTRAT - save strategy

c. TS - terminal support package

d. DBSETU - set information package

e. DBFLDU - field utilities

F. CODING SPECIFICATIONS

1. Source Language

The PRINT command is coded entirely with the IBM PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base, and the TSPL/I extension handles all instances of communication with the terminal.

2. Suggestions and Techniques

a. Normal PL/I statements are used to write the line images to the print data set.

b. The many external variables required in the

PRINT command are combined into external data structures, in many cases. This requires only one name to be an external symbol.

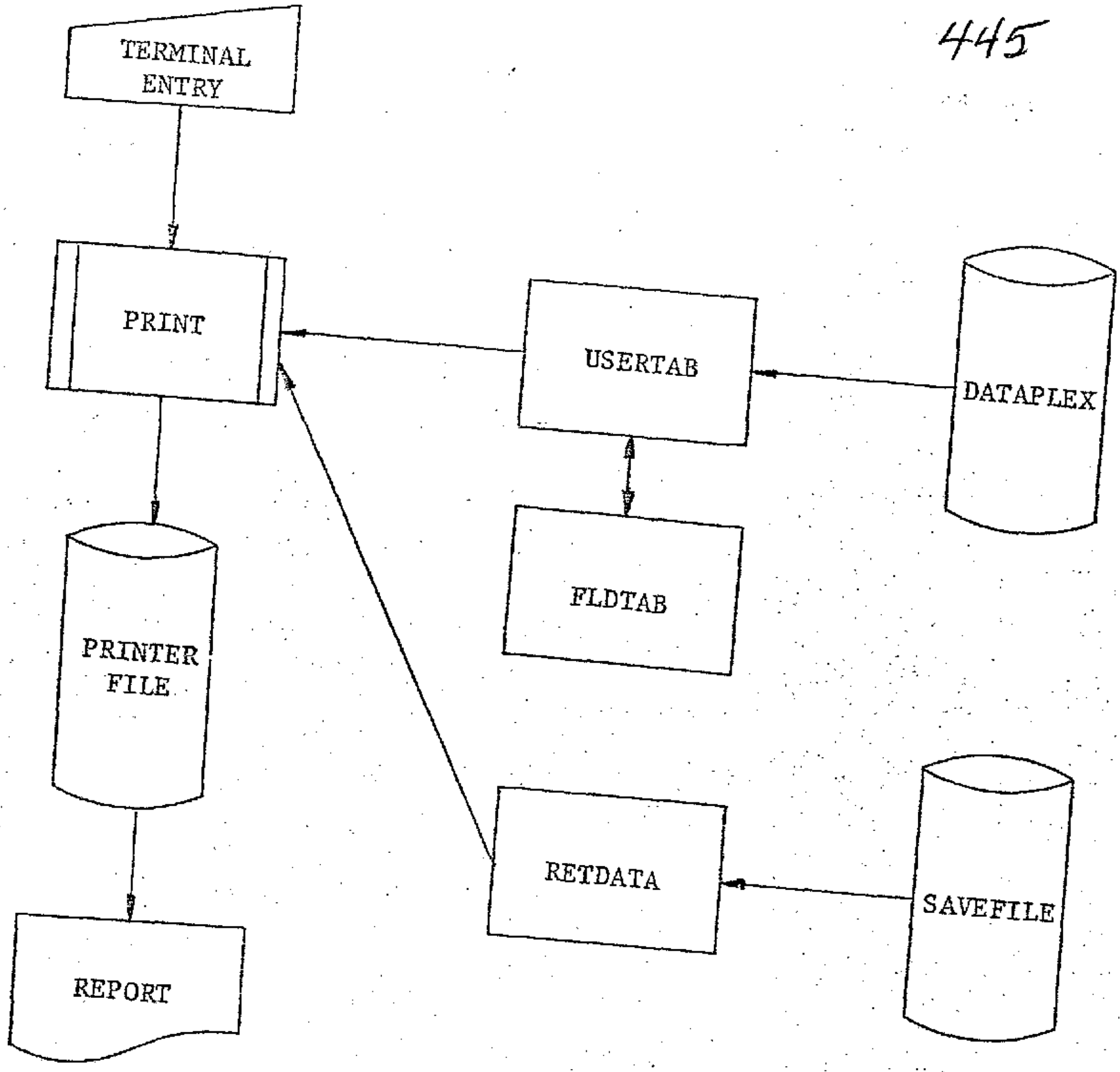


Figure 1. I/O Block diagram

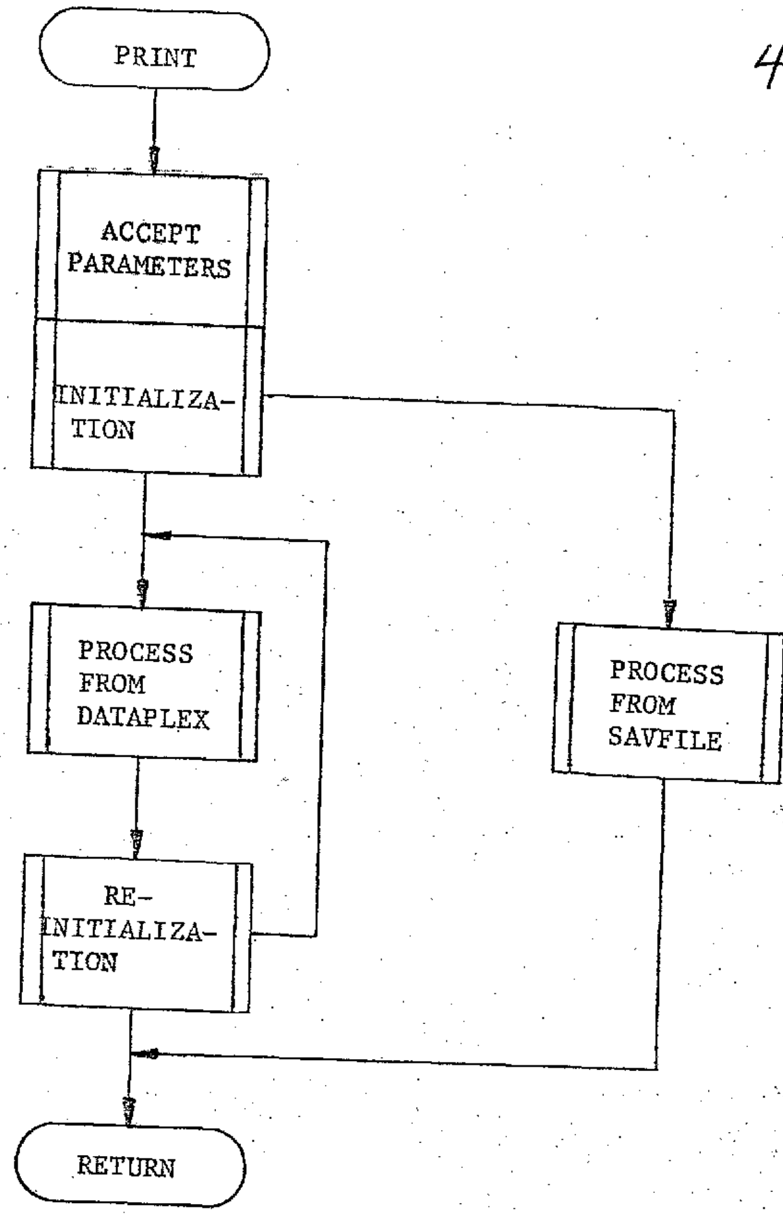


Figure 2. Top level flowchart

TOPIC F.7 - RETRIEVAL EXECUTE COMMAND

A. MODULE NAME

Retrieval, EXECUTE Command
Program-ID - NDBEXSR
Module-ID - DBEXSR

B. ANALYSTS

Barry G. Hazlett
William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

The EXECUTE command's purpose is to identify the overall search universe for all linear search commands defined on the same set, perform a linear search upon that universe, and call, if requested, the SELECT, PRINT, and/or report generator modules to perform set-list formations, list printing or list format printing, respectively. Use of the EXECUTE command informs the NASIS system that user has specified all of his SELECT-IF and/or PRINT commands for his linear search and is now ready to have them executed.

The format of the Execute-Search command is as follows:

EXECUTE

Use of the EXECUTE command informs the NASIS system that the user has specified all of his search requests on one or more sets and is now ready to have them executed. When an attention interrupt is made, the EXECUTE command will return the user with its current status; i.e., the file being searched, the number of processed records and the number of records to be processed. To continue any further in the execution of the linear search, the user must then enter:

GO which will resume the search at the point of execution, or

END which will terminate the search in progress, returning the user to the point of his strategy immediately before the last EXECUTE.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input files

      The dataplex anchor file is accessed to obtain the records for the linear search. The complete description of the files in a dataplex is found in the Data Set Specifications Section of the Workbook.

   d. On-Line Terminal Entries

      If a Terminal is the source of EXECUTE parameters.

3. Output Data Sets

   a. Output Files

      Using the PSTRAT routine, the command string, as it is entered (modified if any by prompt responses) and validated, is saved in the region CURRENT-STRATEGY of the strategy library. For a complete description of the library, refer to the Specifications for the strategy file(DWB, Section IV, Topic H.2).

   b. On-Line Terminal Displays

      The following is displayed at the output interface by the set utilities module (DBSETU):

      1. new set number,
      2. items contained in a new set, and
      3. the (combined) expression describing the new set.

      for each set created as the result of the linear search. All diagnostic messages are displayed in the prompt area of a screen.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. FLDTAB is the descriptor field table referenced to determine the data base name

b. SRCHTAB is the search table referenced to obtain search status switches and pseudo-set table (ENTRYDEF) pointers.

c. Other search tables referenced are ENTRYDEF, S#ENTRY, VALUTAB, VALUE, and SPRNTAB. The data in these tables is described in their respective Data Set Specifications.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The EXECUTE calling sequence is as follows:

CALL DBEXSR

Search processing will follow the following steps:

1. Notify STATISTICS of the search and what dataplex.

2. Call DBSLCT to set up search tables.

3. Identify a search set; group tests on that set.

4. Read in records of the the search set one at a time.

5. For each record test each field against its corresponding test criterion as defined in the LS strategy.

6. Each successful record is added to a

search list associated with the pertinent test pseudo-set.

7. After all records have been tested, new sets are made with the lists for pseudo-sets involved in the search and dependent pseudo-sets defined by a Booolean SELECT via a call to a special entry point in the DBSLCT module.

8. If there is another set to search continue at step 2.

9. All pseudo-sets requiring a "PRINT" are printed via a call to the PRINT command (DBPRNT).

At the search termination all unnecessary dynamic storage will be freed. In addition a special entry point parameter value for the CANCEL SEARCH command will accomplish the same function.

F. CODING SPECIFICATIONS

1. Source Language

The EXECUTE command is written in the IBM PL/I programming language. The DBPL/I and TSPL/I language extensions are used for dataplex file accessing and terminal communication, respectively.

2. Suggestions and Techniques

It is suggested that considerable analysis be made of search universes to determine the final search universe for the EXECUTE command due to the rather large dataplexes that may exist. The success of reducing a search universe to its minimal size is reflected to the user in response time.

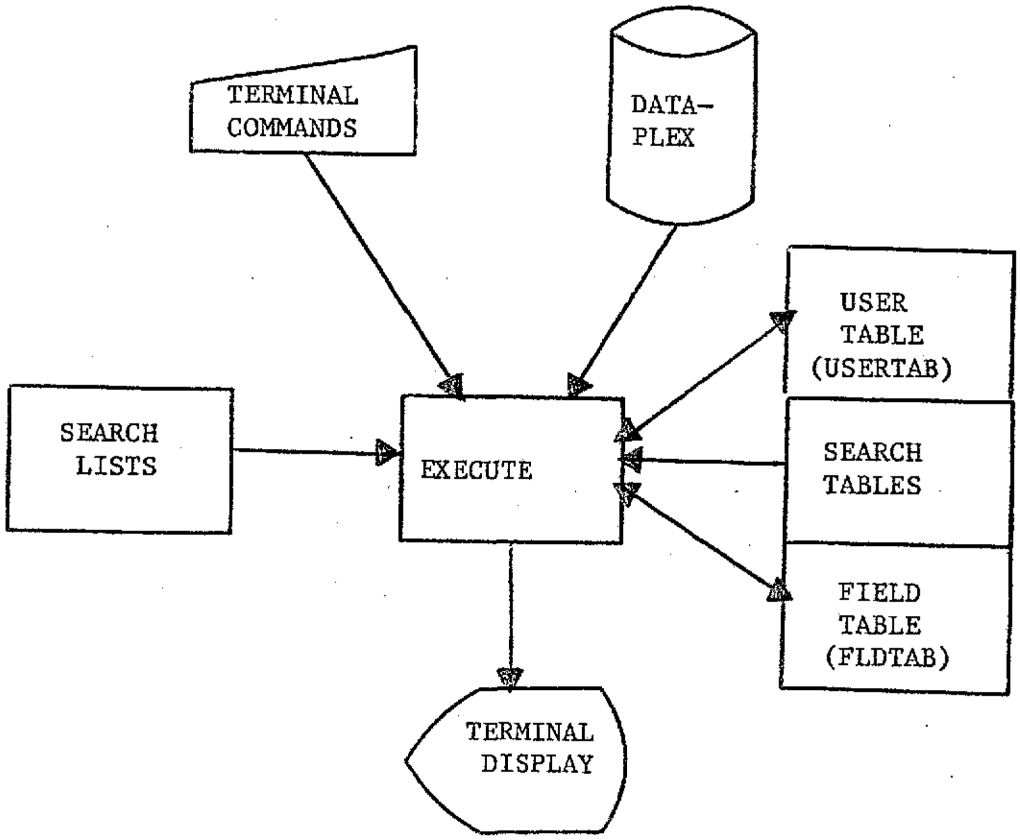


Figure 1. I/O Block Diagram

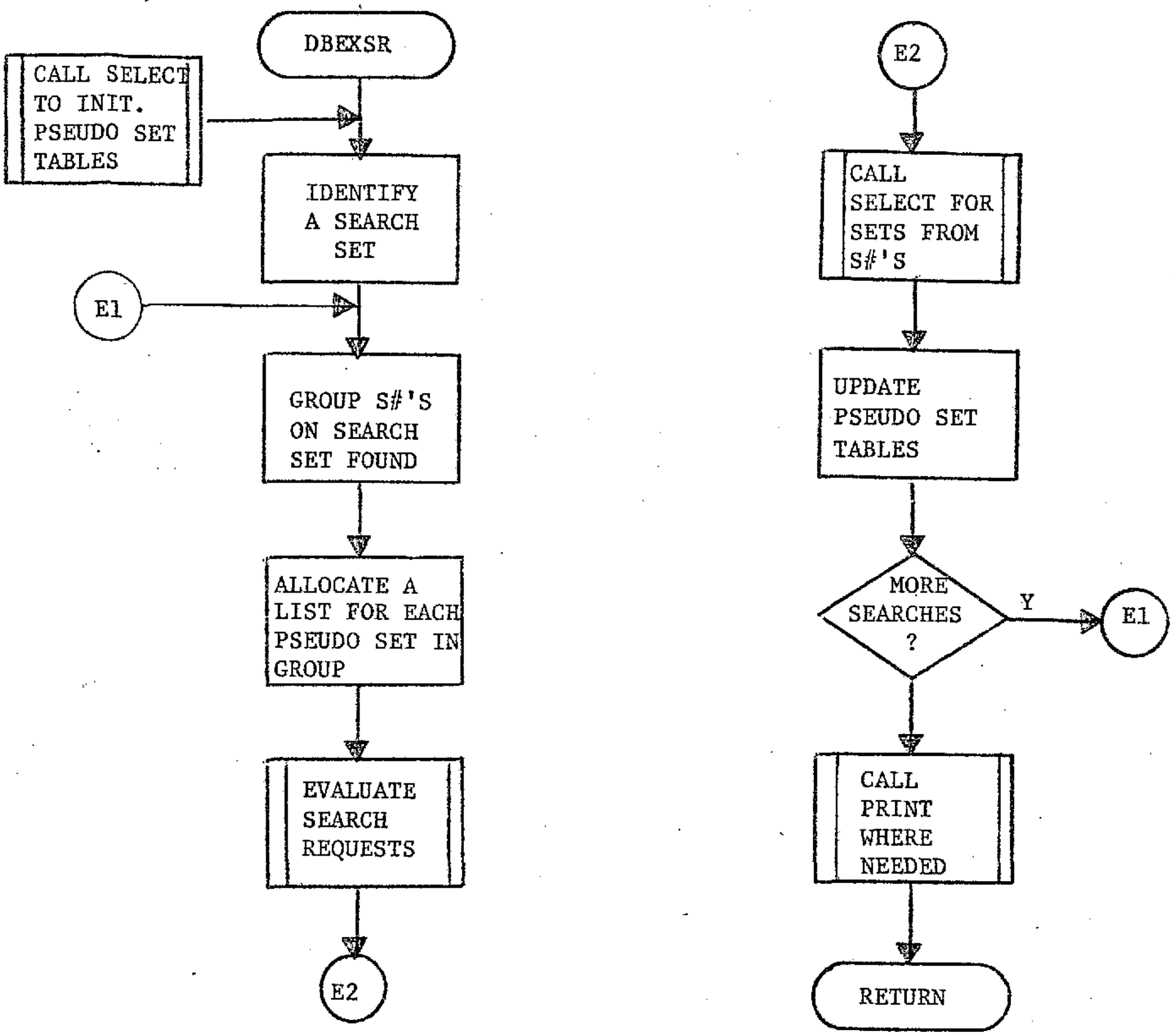


Figure 2. Top Level Flowchart

TOPIC F.8 RETRIEVAL SETS COMMAND

A. MODULE NAME

SETS Command
Program-ID - NDBSETS
Module-ID - DBSETS

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

The primary function of the DBSETS module is to display to the NASIS Retrieval Sub-system user a list of the sets or s-numbers he has formed during the current strategy session. The list is displayed in the form; set number or s-number, number of items in the set, and the expression that formed the set.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      SETAB and Strategy library

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

b. On-line Terminal Displays

The terminal display from this module will consist of a list of the set numbers or s-numbers, the number of items in the set and the expression that formed the set.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. SETAB

b. TS

c. DB

d. Strategy library

e. SRCHTAB, ENTRYDEF, and S#ENTRY

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBSETS

The SETS command is used by the Retrieval System user to display all the sets or s-number (pseudo-set) he has created. Upon entry at the DBSETS entry point, it checks the parameter for paging request; if paging is needed, processing continues at label DBPAGST (see below). Otherwise, it allocates controlled area for the current user to keep track of his paging operations.

The module looks for any parameters that were passed with the command. If there are none, the module will default to set numbers and start displaying at the beginning of SETAB.

If a number between 1 and 97 is passed, the module verifies that as a valid set number and starts the displaying with that number. Paging backwards from this point is not supported, the user need only restarts SETS at a lower set number to achieve this effect.

If the parameter is an 'S' the module will display s-numbers (pseudo-sets). A second parameter may be included here to indicate starting at a specific s-number. Here again, the number is verified, and backwards paging is not supported.

This processing continues until the bottom of SETAB or SRCHTAB is encountered or the TS supervisor indicates the output screen is full and automatically writes the screen.

DBSETS saves the set number or s-number, that would have caused the screen overflow, in the user control table. This set number or s-number is then used as the first number to appear on the next page forward.

b. DBPAGST

This label is called by the TS supervisor when the user wishes to page in either a forward or backward direction through his list of sets.

DBPAGST validates the command and (re)constructs a page in the appropriate direction. Only the letter 'B' will cause a backwards page operation; anything else defaults to forward.

F. CODING SPECIFICATIONS

1. Source Language

The module is written in the IBM PL/I programming language. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Techniques

Not Applicable

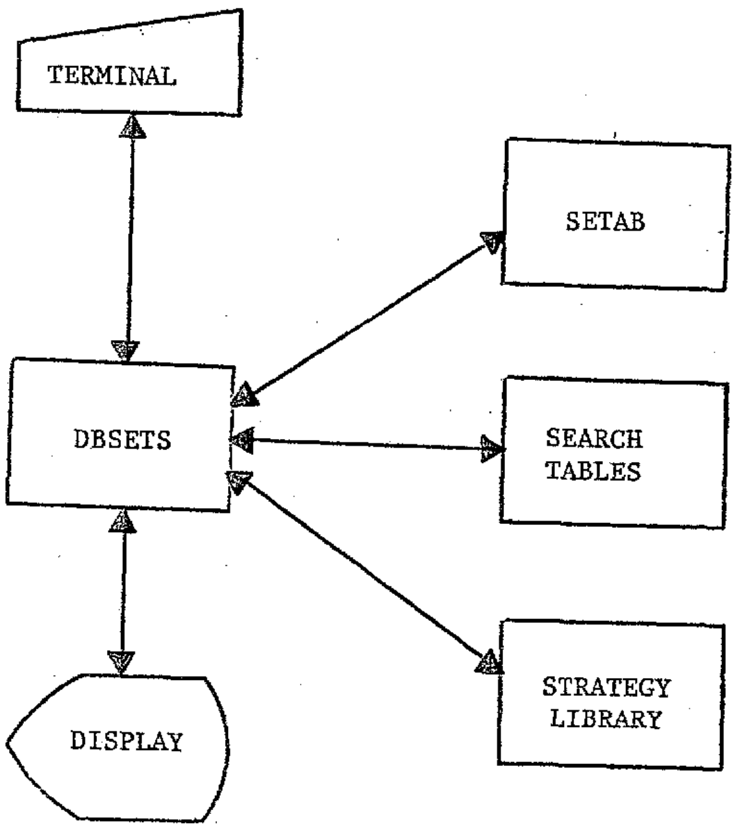


Figure 1. I/O Block Diagram

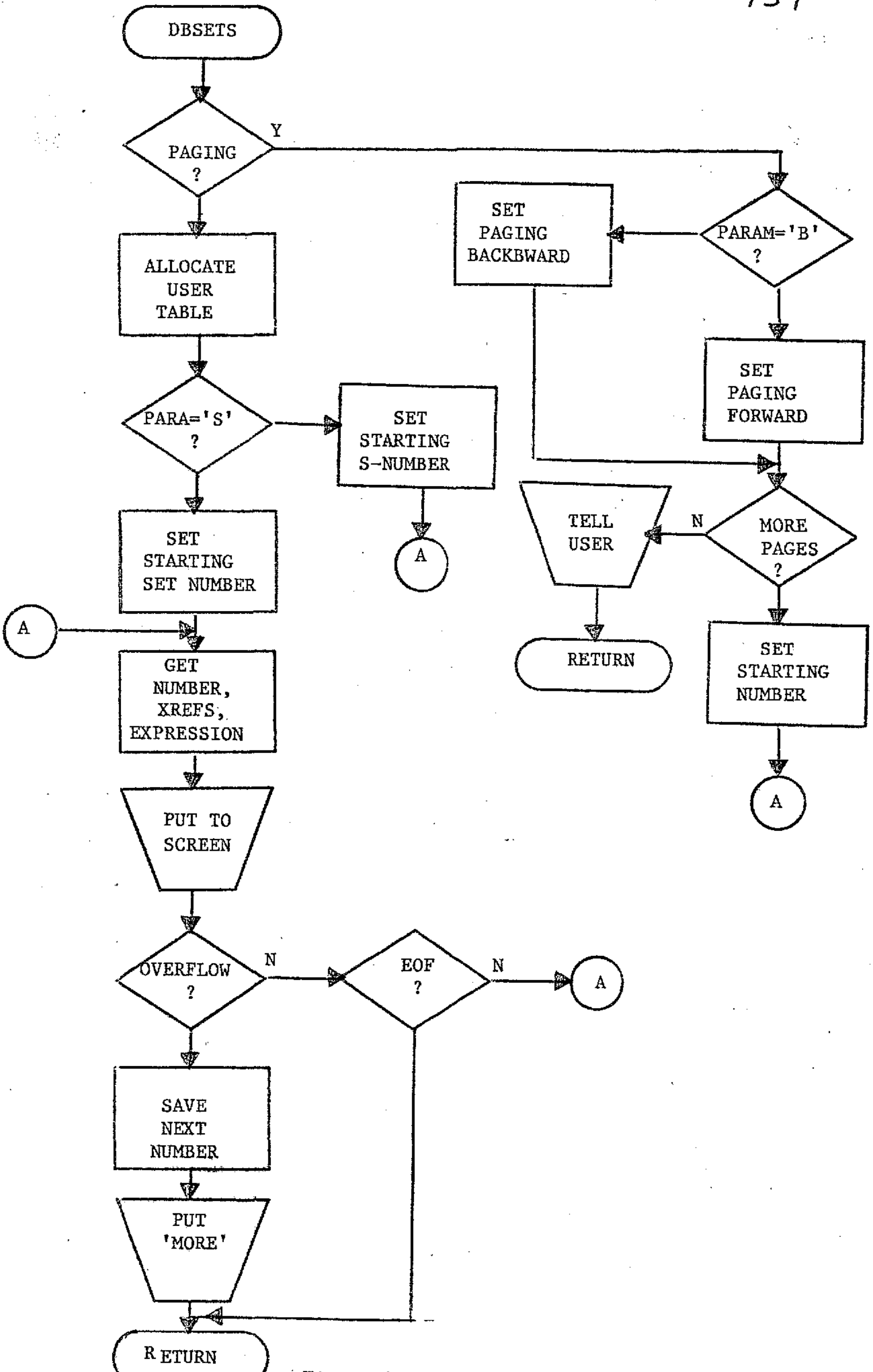


Figure 2 Top Level Flow

TOPIC F.9 - RETRIEVAL SETS UTILITIES

A. MODULE NAME

Program-ID - NDBSETU
Module-ID - DBSETU
Entry Points - DBGSET and DBPSET

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

The function of the DBSETU module is to provide set expression updating functions for various retrieval modules which create of reference sets.

The entry points DBGSET and DBPSET are called from application programs to GET SETS and POST SETS, respectively.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      SETAB and Strategy library

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      SETAB and Strategy library

b. On-line Terminal Displays

The terminal display from this module will consist of a list of the set numbers or s-numbers , the number of items in the set and the expression that formed the set.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. SETAB

b. TS

c. DB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBPSET

This entry point is available to the application programmer who wishes to post a new set and its corresponding data to SETAB. The calling sequence follows:

CALL DBPSET(POINTER,EXPRESSION,SET#);

Where:

POINTER - is a pointer variable passed by the user. It points to the list to be posted.

EXPRESSION - is a varying length character string, maximum 256 bytes. It is passed by the user as the expression that formed the set to be posted.

SET# - is a varying character string, maximum 2 bytes long. It is passed by the user as the one byte subfile suffix character for the

set being posted and is returned by DBPSET as the 2 byte set number on a successful posting or a null string to indicate an I/O error or no more sets available.

This entry point first checks for a slot in SETAB; if none are available, it sets the set number variable to null and returns to the user.

If a set number is available,it verifies the suffix as being between Q and Z or it assigns a blank suffix. DBPSET then collects and posts the data to SETAB and the STRATEGY LIBRARY. It posts the set number for the user and returns.

b. DBGSET

This entry point is available to the application programmer who wishes to get and verify a given set number. The calling sequence follows:

CALL DBGSET(SET#,POINTER,#LIST,SUFFIX);

Where:

SET# - is a varying length character string, maximum 3 bytes long. The user passes this variable as the set number, and optionally the subfile suffix, to be gotten and verified. If either the set number or the suffix is invalid, that is, a non-existent set number or a wrong suffix, this variable is returned as null.

POINTER - is a pointer variable. It is returned by DBGSET as a pointer to the set (list).

#LIST - is a integer full word. It is returned by DBGSET as the number of XREFS in the set.

SUFFIX - is a single character. It is always returned as the correct suffix for the set requested. In the event an invalid suffix is specified in the set number, the set number is returned as null and the correct suffix is returned here.

DBGSET first separates the set number from

the suffix and verifies both. If either is invalid, set number is returned as null and the correct suffix, if available, is put in SUFFIX. If the validation is successful, the set number, the list pointer, the number of XREFS and the suffix are returned to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written in the IBM PL/I programming language. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Techniques

Not Applicable

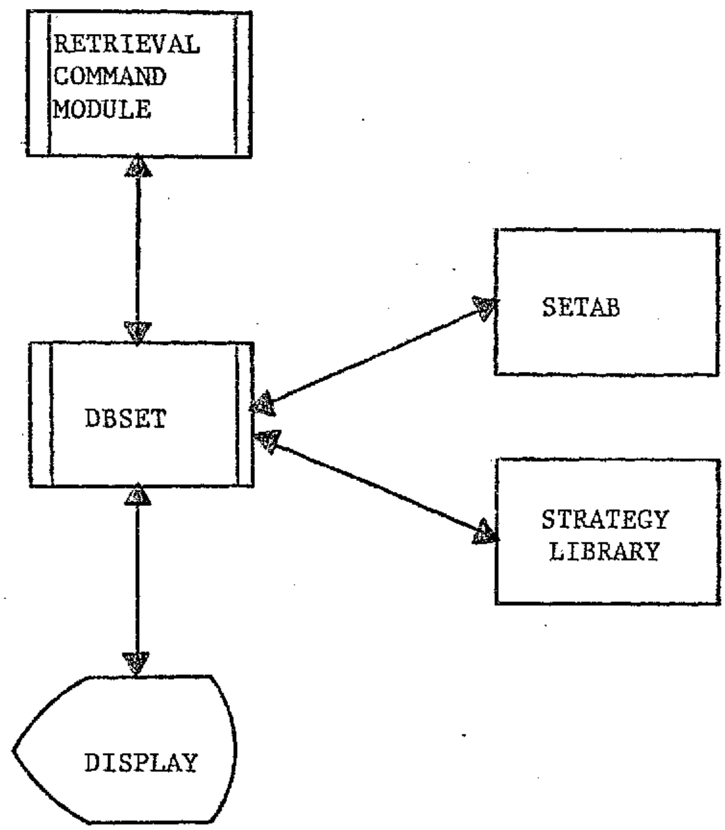


Figure 1. I/O Block Diagram

TOPIC F.10 - RETRIEVAL FORMAT COMMAND

A. MODULE NAME

Retrieval, FORMAT Command (module 1 of 2)
Program-ID - NDBFORM
Module-ID - DBFORM.

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

The DBFORM module is the first FORMAT command routine, called by the retrieval system, whose purpose is to allow the retrieval system user to define, revise and/or display the content and format for subsequent information retrievals using the DISPLAY or PRINT retrieval commands. Sequential and columnar formats may be defined.

Sequential formats extend the series of predefined formats 1-4 by allowing the user to select a set of fields to be displayed one under another with no more than one record's fields per output page.

Columnar formats are a separate series allowing the user to select a set of fields to be displayed in tabular format. Optionally, the user may define screen or printer output, page numbering, titles, column headers, column positions, and element tallying, summing and averaging.

After a current format has been established, the DBFORM module functions as a command director processing the FIELD, FIELDS, NAME, STORE, FORMATS, DISPLAY, PAGE, TITLE, HEADER, FORMAT and END subcommands of the FORMAT command. Although DBFORM recognizes all FORMAT subcommands, for the HEADER and FIELD subcommands a transient call is made via DBINIT to the second FORMAT command module DBFORMA. Refer to DBFORMA Program Design Specification for the details of these two subcommands.

The user may review the appearance of the ultimate display (paging through screen-width portions, if necessary). The user has complete revision and storing capability.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      A terminal is the most likely source of the parameters which are passed to the FORMAT command by the Terminal Support system. The fundamental parameters are the format number and the field names. Default values for the fundamental parameters are unlikely. The FORMAT command then accepts the FORMAT subcommands and their parameters.

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      For sequential formats, the DISPLAY subcommand will display the field names vertically in the order they will ultimately be displayed. The PAGE subcommand will display any field names that do not appear on the first screen.

      For columnar formats, the DISPLAY subcommand will display the title and header values and field column positions as they will ultimately be displayed. In the case of printer formats wider than the display screen, the left-most portion will be displayed initially. The PAGE sub-command will display subsequent portions. These displays will show the positioning and length

of the field values for the first data line; otherwise, they have the same format as the DISPLAY and PRINT retrieval commands produce (see Section III, Topic F.4 of the DWB).

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COL_FORM

When the FORMAT command processes a new columnar format, it allocates and initializes a COL_FORM structure and posts its base address in the COL_FORMAT array in FLDTAB. When the FORMAT command processes a TITLE or HEADER sub-command or any other revision to a columnar format, it updates the appropriate COL_FORM structure. Thus, a COL_FORM structure specifies a columnar format for use by the DISPLAY and PRINT commands.

b. FLDTAB

The FORMAT command refers to DATAPLEX portion of FLDTAB. The FORMAT command also posts the SEQ_FORMAT and COL_FORMAT arrays as it processes new formats.

c. SEQ_FORM

When the FORMAT command processes a new sequential format, it allocates and initializes a SEQ_FORM structure and posts its base address and field name count in the SEQ_FORMAT array in FLDTAB. Thus, a SEQ_FORM structure specifies a sequential format for use by the DISPLAY and PRINT commands.

d. USERTAB

The FORMAT command checks the USERTAB.RETRIEVE switch to verify that it is being called properly.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   a. Format

      The FORMAT command is recognized by the retrieval system director module DBINIT which calls the DBFORM entry point.

   b. Process FNUMBER parameter

      An FNUMBER parameter value is obtained from the Terminal Support system. If null or blanks are entered, the FORMAT command is cancelled. The value is checked for proper syntax and for range and duplication of the number; errors are diagnosed and the user allowed to re-enter. If the value is a name, the external GETSFMT routine is called to obtain the stored format. For a new format, a SEQ_FORM or COL_FORM structure is allocated and initialized according to given and default options and the structure's base address posted in FLDTAB. For a revised columnar format, any options given will result in the COL_FORM structure being modified or re-allocated and initialized accordingly. Removal of page numbering may be specified and/or expansion to printer width or contraction to screen width. If the width changes, any titles are re-centered. If the width changes and the columns are proportional, they are re-proportioned and their headers (if any) re-centered. If the width expands and the columns are explicit, the rightmost column will have its width expanded and its headers (if any) re-centered. If the width contracts and the columns are explicit, columns to the right of a screen width are dropped from the format with their headers (if any) and the remaining rightmost column will have its width reduced and its headers (if any) re-centered. Thus, a current format has been established for further processing.

      If a FLDSPEC parameter was entered explicitly by the user with the FORMAT command, control passes to (d.) below where the parameter is processed. Otherwise, processing continues

at (c.).

c. Process subcommand

A command is obtained from the Terminal Support system. If it is a valid FORMAT subcommand, it is processed by one of the routines (d.) through (k.) below. Otherwise, it is diagnosed as an invalid subcommand and the user allowed to re-enter.

d. Process FIELD command

This subcommand is handled by the DBFORMA module; it is called via the DBINIT Transient Module Interface convention. Necessary information is retained in a common structure called FORMCTL.

e. Process FIELDS or FORMATS command

These commands are recognized as a convenience to the user to save him having to leave FORMAT and later re-enter it. Processing consists only of a call to the external entry point DBFLDS or DBSTRT2 respectively; the DBFLDS module is called via the DBINIT Transient Module Interface convention.

f. Process NAME or STORE command

An FMTNAME parameter value is obtained from the Terminal Support system, validated syntactically by calling the external DBUCHEK routine, and checked for duplication of the name of any other current format. For a NAME command, the value is simply posted in FLDTAB. For a STORE command, the value is posted in FLDTAB or it is verified that a name value was posted there previously and the external PUTSFMT routine is called to store the format for availability in later sessions. If the FMTNAME value is invalid or missing or if PUTSFMT returns an error code, a diagnostic is issued and the user allowed to re-enter it.

g. Process TITLE command

If the current format is not columnar, the TITLE command is cancelled with a diagnostic message.

A TTLLINE parameter value is obtained from the Terminal Support system, if the user entered it explicitly, or by assuming the next relative title line number. The value is checked for syntax, range, duplication, and space in COL_FORM.TOP. Any error is diagnosed and the user allowed to re-enter the parameter. For a title line deletion, any lower title and header line images are shifted up and COL_FORM.TOP.#TITLES is decremented and control branches to (c.). For a new title line, any lower title and header line images are shifted down and intervening lines blanked in COL_FORM.TOP.LINE and COL_FORM.TOP.#TITLES is posted.

A TTLSPEC parameter value is obtained from the Terminal Support system, if the user entered it explicitly, or by taking the FLDTAB.DATAPLEX name value and stripping any trailing dollar sign characters. The value is posted centered in the particular COL_FORM.TOP.LINE.

h. Process HEADER command

If the current format is not columnar, the HEADER command is cancelled with a diagnostic message.

This subcommand is handled by the DBFORMA module; it is called via the DBINIT Transient Module Interface convention. Necessary information is retained in a common structure called FORMCTL.

i. Process DISPLAY command

A display screen image is composed and transmitted via the Terminal Support system to the user's terminal showing the format as currently defined. The display simulates the appearance produced by the retrieval system DISPLAY command if it was used with the current format.

If a sequential format display overflows the screen at the bottom or if a columnar format display overflows the screen at the right side, "MORE" is indicated and the Terminal Support system is requested to call DBFORMP if the user enters the PAGE immediate

command.

When the module is entered at the main entry point with a parameter value set for paging, a DIRECTON parameter value is obtained from the Terminal Support system, if the user entered it explicitly, or by assuming forward paging. If the value starts with "B" the previous display screen image is re-composed, otherwise the next display screen image (down or to the right) is composed. Screen overflow is rechecked to reset the "MORE" indication and the Terminal Support system transmits the screen image to the user's terminal.

j. FORMAT command

If "FORMAT" is detected as a sub-command, control simply branches up to (b.) where its parameters are obtained and it is processed.

k. END command: RETURN

The END subcommand causes control to be returned to the retrieval system director module DBINIT.

If the END condition is raised by the user entering the END immediate command in blocks (a.) or (b.), control returns to the DBINIT module. If it is raised after block (b.) control branches to block (c.), that is, the subcommand is aborted and another taken.

3. Submodules required

DBFORMA - FORMAT command, module 2
DBFLDU - field utilities
DBFLDS - FIELDS subcommand
DBSTRT2 - FORMATS subcommand
DBUCHEK - check name routine
GETSFMT - get stored format
PUTSFMT - put stored format
PSTRAT - save strategy
TS - terminal support package

F. CODING SPECIFICATIONS

1. Source Language

The FORMAT command is coded in IBM PL/I. The

TSPL/I language extension is used for all communication with the terminal.

2. Suggestions and Techniques

The PSTRAT external routine shall be called whenever a valid command or subcommand with valid parameters is detected.

Subroutine facilities shall be coded to handle the general case of re-proportioning columns, re-centering headers, and shrinking sequential formats (DUP_COL, RE_PRO_COL, RE_HEAD, SHRINK_SEQ).

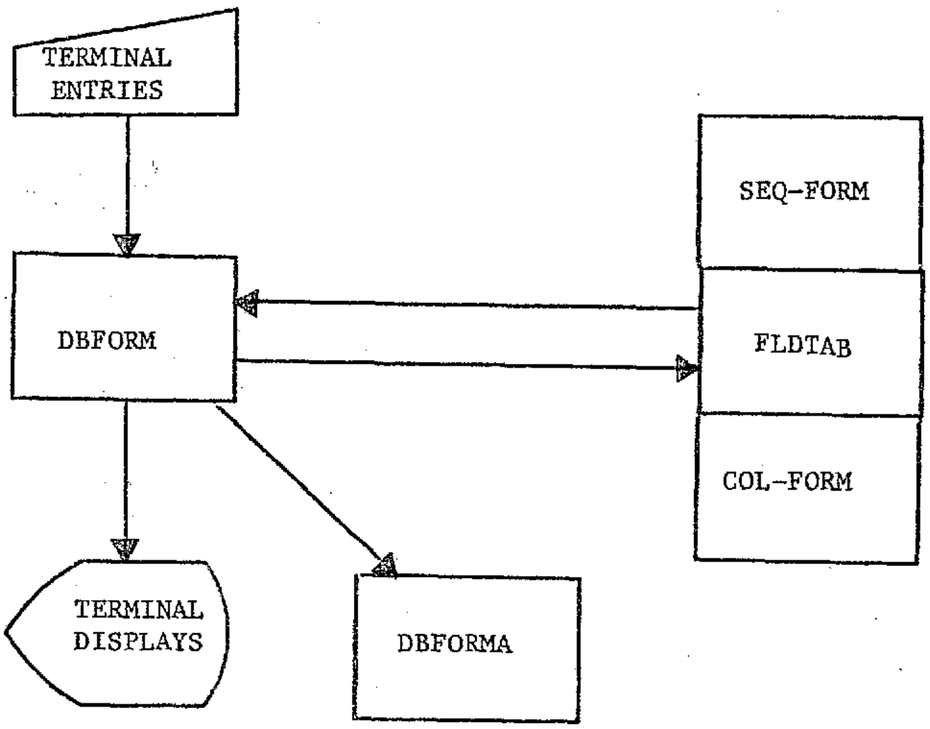


Figure 1. I/O Block Diagram

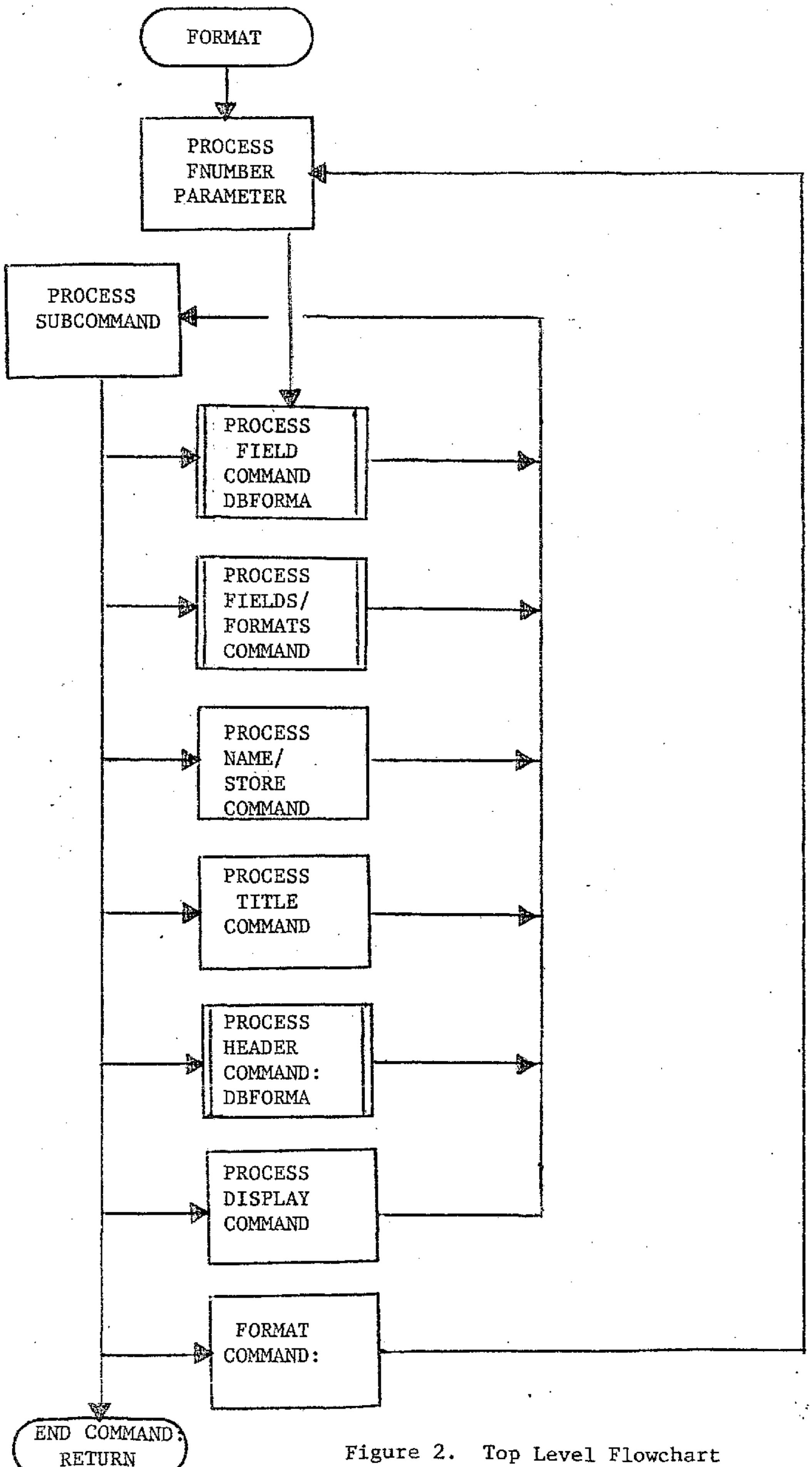


Figure 2. Top Level Flowchart

TOPIC F.11 - STORED FORMATS

A. MODULE NAME

Program-ID - NDBSFMT
Module-ID - DBSFMT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The function of this module is to provide generalized GET/PUT routines for the processing of stored formats.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input File

Not Applicable

c. Input Files

Not Applicable

d. On-time terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces diagnostic messages for the various errors that may occur.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The following tables are referenced, used in the construction of new formats and used to output exiting formats.

FLDTAB
SEQFORM
COLFORM

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. GETSFMT

At this entry point the program initializes itself to read in a previously stored format. It verifies the name of the format and checks to see if the format is already in the format table. If so, the program returns immediately with the appropriate information.

If the format must be reading, the first record of the format is obtained by calling TSGETRG. This record is analyzed to determine if the format is columnar or sequential. The appropriate format tables are then searched for a slot into which the format can be placed and the format is allocated and initialized.

The program then obtains the remaining format records and posts the data obtained into the appropriate lccations within the format entry. If any errors are encountered, an appropriate diagnostic message is written to the user and the partial format is freed. After an error, or when the format has been completed, the required information is

updated and the program returns to the caller.

b. PUTSFMT

At this entry point the program initializes itself to write out one of the currently defined formats. It verifies the name of the format and checks to see of the format exists in the format tables. If not, the program terminates with a diagnostic.

If everything is in order, the program constructs the first format record (FORMAT), indicating the format name, type, the intended file name and other descriptive information and writes it to the data set by calling TSPUTRG.

The remaining format data is organized into TITLE, READER and FIELDS records and written to the data SET in the same fashion as the FORMAT record. If any errors are encountered, an appropriate diagnostic message is written to the user and the partially stored format is erased. After an error, or when the format has been completely written out, the required information is posted and the program returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the IBM PL/I Language.

2. Suggestions and Techniques

Not Applicable

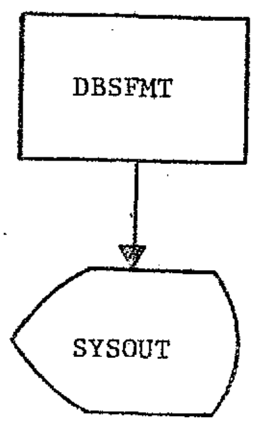


Figure 1. I/O Block Diagram

TOPIC F.12 - E NUMBER ROUTINE

A. MODULE NAME

Program-ID - NDBXPNDE
Module-ID - DBXPNDE

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module is called by DBSLCT to validate an E-number and, if valid, to return associated data.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   Not Applicable

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      Not Applicable

   c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

EXPTAB
EXPTERM

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

This module initializes itself to decode an E-number reference. If the E-number parameter is valid, the data associated with it is passed back to the caller (DESLCT) and the program is terminated.

F. CODING SPECIFICATION

1. Source Language

The module is written in IBM PL/I language.

2. Suggestions and techniques

Not Applicable

TOPIC F.13 - BATCH PRINT MONITOR

A. MODULE NAME

Program - ID - NDBPRINT
Module - ID - DBPRINT

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTION

This program controls the execution of the batch print system in much the same way that NDBINIT controls the retrieval system. That is, it initializes file-related tables and issues command prompts to activate batch sub-system operations.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

NASIS.USERIDS

d. On-line Terminal Entries

The user of the batch print system communicates with the system through a series of command and data prompts. The commands and parameters are:

1. END

Terminate the terminal session

2. PRINT NASISID=,BSN=

Produce a formatted print-out of data from a file utilizing information saved in the print queue for Nasis ID with Batch Sequence Number (BSN) specified.

3. HOLD NASISID=,BSN=

   Place a print job in "hold" status.

4. RELEASE NASISID=,BSN=

   Place a print job in "active" status so that it can be executed.

5. EXHIBIT NASISID=,BSN=

   Display a formatted description of the contents of the batch print queue at the user's terminal.

6. NUMBER NASISID=

   Tally the number of print tasks in the queue.

7. CANCEL NASISID=,BSN=

   Remove a print task from the queue.

8. KEYS NASISID=,BSN=

   Display the file name and record keys recorded for a print task.

9. COPIES NASISID=,BSN=,COPIES=

   Overide the user specified value for number of copies of a printed report.

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      Not Applicable

   c. Formatted Print-outs

      Not Applicable

   d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

See Figure 2.

2. Narrative

DBPRINT gets control from DBMTT, then prompts for one of the commands outlined in section 2D. If the command is PRINT, the information relating to the user's print queue is retrieved from the strategy data set and used to open the file from which data is to be printed. After all initialization is complete, control is passed to DBWRIT to perform the actual data retrieval and printing.

All other commands provide various operations on the user's print queue as described above, except END, which returns control to DBMTT.

F. CODING SPECIFICATIONS

1. Source Language

PL/I

2. Suggestions and Techniques

Not Applicable

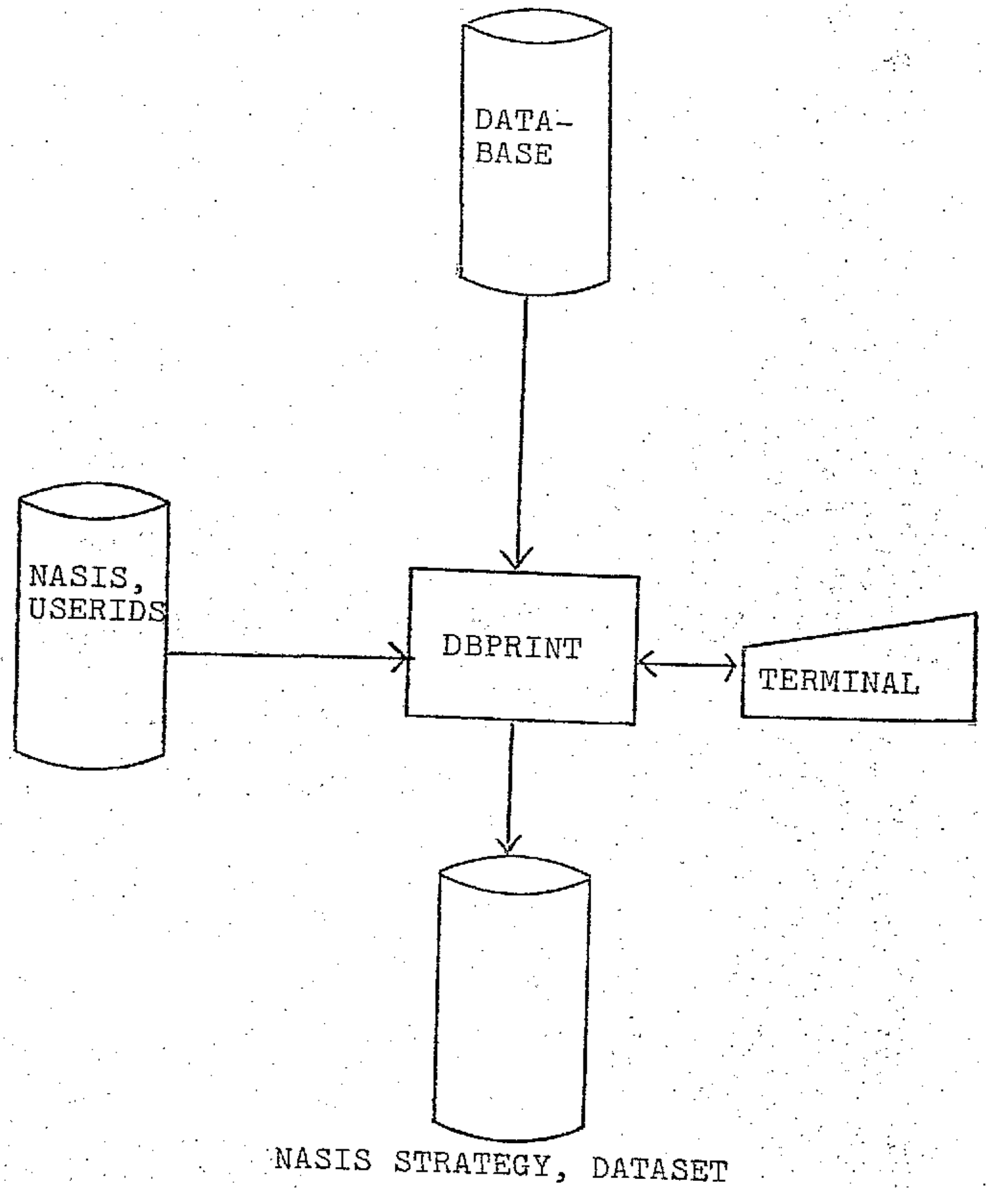


Fig. 1 I/O Block Diagram

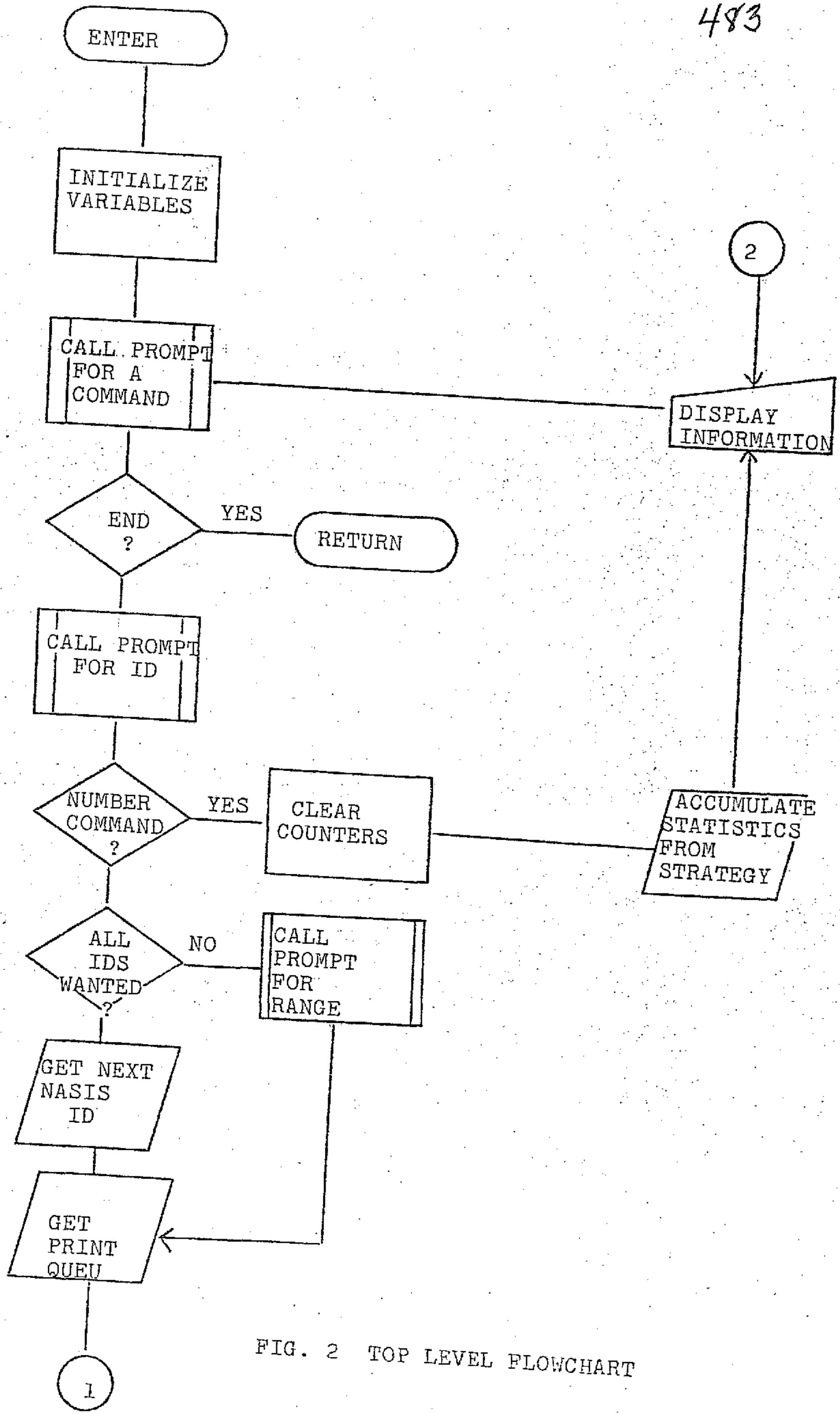


FIG. 2 TOP LEVEL FLOWCHART

TOPIC F.14 - BATCH PRINT WRITER

A. MODULE NAME

Program - ID - NDBWRIT
Module - ID - DBWRIT

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTION

This program retrieves data from a user - specified data base and prints a listing in either a predefined sequential format or a user-defined sequential or columnar format.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Any NASIS data base.

d. On-line Terminal Entries

None

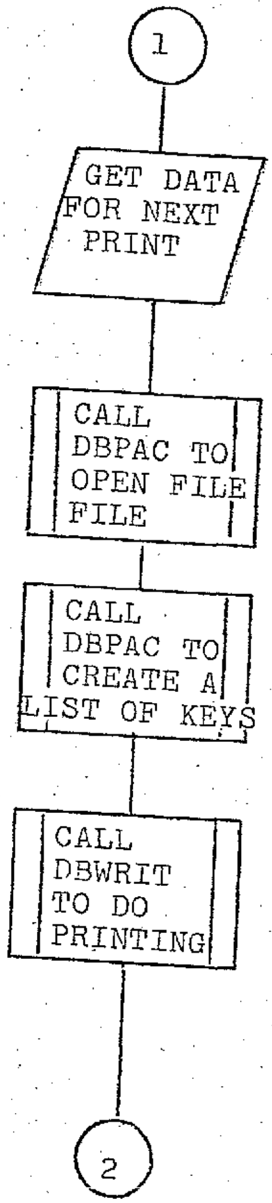
3. Output Data Sets

a. Output Files

Print file (PRINTER)

b. On-line Terminal Displays

Not Applicable


1
GET DATA FOR NEXT PRINT
CALL DBPAC TO OPEN FILE FILE
CALL DBPAC TO CREATE A LIST OF KEYS
CALL DBWRIT TO DO PRINTING
2

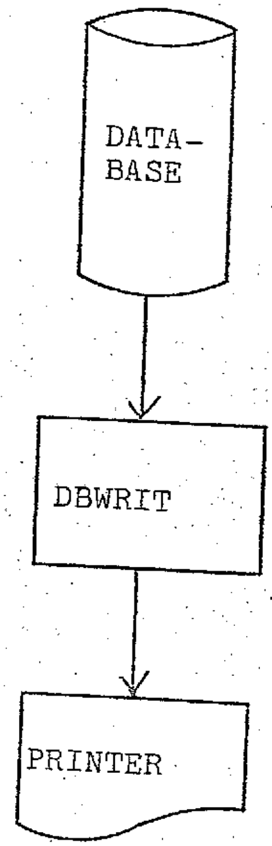


FIGURE 1 I/O BLOCK DIAGRAM

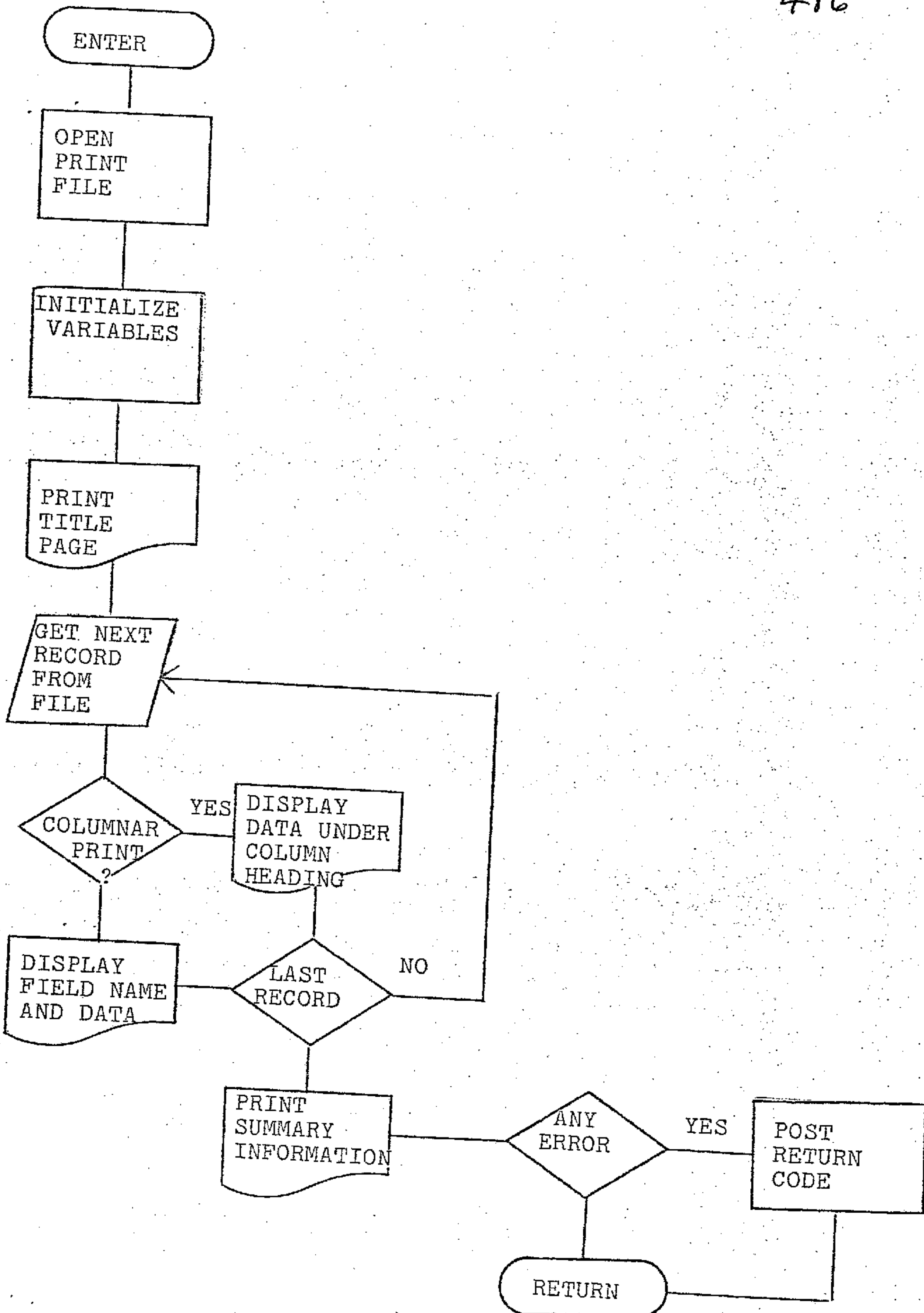


FIGURE 2 TOP LEVEL FLOWCHART

c. Formatted Print Outs

User - defined sequential or columnar prints.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

See Figure 2.

2. Narrative

DBWRIT gets control from DBPRINT, then opens the PRINTER output file and creates the title page. Next, a record from the data base being retrieved from is read and either sequential or columnar formatting is begun based on a table of field names specified by the user. For sequential formats, the field names and associated data are displayed on successive lines with the field names to the left of the data. Columnar formats require the printing of header and title information (saved by the PRINT and FORMAT functions) along with the field names or other identifier for each column of data across the top of each page. The data for each field is presented under the appropriate column heading until the list of record keys is exhausted.

When all printing of data is completed, a summary of information contained therein is displayed. For sequential prints this is simply a count of the number of records displayed. For columnar prints, this can be, optionally, a tally, sum, and average of the numerical values of items occurring in one or more of the columns.

After closing the PRINTER file, control is returned to DBPRINT with a return code of 'X' for a print terminated by the operator of 'O' for a print terminated by a data base error. The return code is unchanged if the print completes successfully.

F. CODING SPECIFICATIONS

1. Source Language

   PL/I

2. Suggestions and Techniques

   Not Applicable

TOPIC F.15 - RETRIEVAL CORRECT COMMAND

A. MODULE NAME

Retrieval, CORRECT Command
Program-ID - NDBCORR
Module-ID - DBCORR
Entry Point (DBCORR)

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The CORRECT command is a routine, called by the RETRIEVAL system, whose purpose is to allow the retrieval system user to create certain maintenance transactions during retrieval. When a user observes an error during a display, he is able to have any or all of the fields of a given record displayed and then be able to specify any deletions, additions, or changes to those fields. The transactions created are not executed, but are placed in a transaction data base which is examined by the data base owner before the actual maintenance takes place. The calling sequence is: CORRECT field,key.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      A terminal is the most likely source of the

parameters which are passed to CORRECT by Terminal Support. The parameters available to the CORRECT command are "key" and "fieldname". The Terminal Support system applies default values to the parameters, if they are available, when no original values are entered.

Additional terminal entries are requested of the user. These responses indicate what alterations, if any, to the field are desired. These entries take the form of sub-commands available to the user while running under control of CORRECT. The sub-commands are:

ADD data
CANCEL
CORRECT field,<key>
DELETE element
DISPLAY
END
FIELDS
INSERT field,...
REPLACE element1,<element>,olddata<,newdata>
VERIFY

3. Output Data Sets

a. Output Files

The only output file from the CORRECT command is the transaction data base. This file is a QISAM data set containing maintenance transactions from all sources for all data bases. The fields of the transaction data base and their format are completely described in the Dataset Specifications.

b. On-line Terminal Displays

The CORRECT command outputs a formatted display of the specified field on the terminal. Each field to be processed begins on a new screen image with appropriate header information. Each element of multi-element fields begins on a new line. No attempt is made to end lines of the display on word boundaries. In addition to the display of the field in question, a prompting message requesting the action to be taken is issued in the input area of the screen.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROGRAM REQUIREMENTS

1. Flowchart

See Figure 2

2. Narrative

a. CORRECT

The CORRECT command is called by the retrieval sub-system at entry point DBCORR. Any default paramenters which are applicable are supplied by terminal support.

b. Real Entry

This routine initializes the routines for handling the exceptional error and interrupt conditions. Attention interrupts cause the user to be prompted for a decision. If he defaults, execution continues from the point of interruption. Terminal support prompting errors cause program termination, unless the error is for input transaction, in which case, a warning message is issued to the user and execution continued. Any other errors cause program termination, following an appropriate diagnostic message.

c. Main Line

The routine allocates the screen buffers, if not already done, and obtains the julian date for time stamping the transactions. The current retrieval data base is then opened for input, unless that data base is the user's transaction data base, in which case, it is opened for update. The parameters passed by the user are validated. The user is prompted for a field name if he failed to

enter it initially.

d. Get Record

If necessary, this routine reads a new record from the input data base and gets the value of the key field, the latter necessitated by the optional sequential mode of operation. Again, if necessry, the routine reads in the values for all elements of the field, maintaining a count of the number of elements processed. Finally, the routine copies the input data to a temporary storage area for the user to process against.

e. Format Screen

This routine, unless running from a typewriter with the verify option equal to no, formats and displays the status of the data most recently referenced by him. It first constructs a heading, composed of the record key, data base name, field name and element count. It then proceeds to fill the remainder of the screen with data beginning at the element indicated by the calling routine. If the element length is less than the length of the data portion of the line, the element is written on a single line preceeded by the element number. If the element is too large, the first line is processed as above, but the remaining data is split across succeeding lines.

f. Re-prompt

This routine prompts the user for his next request. It extracts the command keyword, and if valid, calls the appropriate calling routine. If any type of error is encountered, the routine re-prompts for the correct information.

g. Add Routine

This routine takes the input data and uses it to create a new element for the field being processed. If FIELD is key field, then new record is created. If there is no data entered, or if the maximum allowable number of elements has been reached, a diagnostic is written and processing bypassed. After processing, that data control is passed to

Format Screen to display the updated data.

h. Replace Routine

This routine expects four parameters to be entered, a starting point, an ending point, an old data value and a new data value. The starting and ending points are expressed in terms of element numbers. If the element numbers are invalid, or if no old data value is entered, a diagnostic is written and processing bypassed. The data is then searched, character by character, from the starting point to the ending point. If any occurrence of the old data value is found, it is replaced by the new data value. If no occurrence of the old data value was found, a diagnostic is written. Otherwise, control is passed to Format Screen to display the updated data.

i. Cancel Routine

This routine re-initializes the data in the field to its initial status when read from the data base. Control is then passed to Format Screen.

j. Page Routine

This routine expects one parameter, which it uses to adjust the current element pointers to adjust the segment of the data which is displayed on the screen. The parameter may be a default for forward paging, a 'B' for backward paging, a number for a specific element number. If the data is invalid or the request cannot be honored, a diagnostic is written and processing bypassed. Otherwise, control is transferred to Format Screen to display the data.

k. Verify Routine

This routine sets the switch that determines whether the user, operating from a typewriter, receives a verification display of the data following each command. The data entered should be 'YES' for verification or 'NO' for none. If the data is invalid, a diagnostic is written and processing bypassed. Otherwise, control passes to Format Screen.

l. Delete Routine

This routine uses the user's data to decide whether to delete the entire record, to delete the field, to delete an element or to delete a range of elements. If the field is to be deleted, it is done and control passed to Format Screen. If the record is to be deleted, it is done and control passed to End Routine. If elements are to be deleted, the routine will accept a list of elements or element ranges as input. For each, it analyses the element number to determine its validity. An invalid element will cause a diagnostic to be written and further processing bypassed.

m. Insert Routine

This routine allows the user to specify the fields of subfile records to be inserted into the dataples. If no field is specified, a diagnostic is written and processing is bypassed. If the previous field's data has been changed, Output Routine is called to create the necessary transactions. Control is then passed to the Correct Routine for further processing.

n. Correct Routine

This routine allows the user to specify the key of a new record to be processed, the name of the next field to be processed, or both. If the previous field's data has been changed, Output Routine is called to create the necessary transactions. The routine first checks for a signed numeric value in the key operand, and if found, reads the file sequentially forward or backward to the desired record. If sequential processing is not indicated, the routine extracts the new key and the new field name, if present, and transfers control to Get Record for further processing.

o. Fields Routine - CALL DBFLDS

This routine displays a list of the field names for the data base for the user. It calls DBFLDS to extract the field names. It also checks each field until it has identified the key field, whose name it

maintains separately. It then moves the field names into the output area, fitting as many as possible on each line, and displays them to the user. If more names exist than may be displayed on the screen at once, the routine prompts the user for a decision as to whether he wants to see the remaining names or to continue correcting.

p. End Routine

This routine processes any transactions remaining to be written. It closes all of the files, resets switches, restores the NASIS status to what it was when the program was invoked and returns to the calling program.

F. CODING SPECIFICATIONS

1. Source Language

The NDBCORR program employs the IBM PL/I programming language. The special extensions of that language, called DBPL/I and TSPL/I, are utilized for all access to files and for all terminal communication, respectively.

2. Suggestions and Techniques

Not Applicable

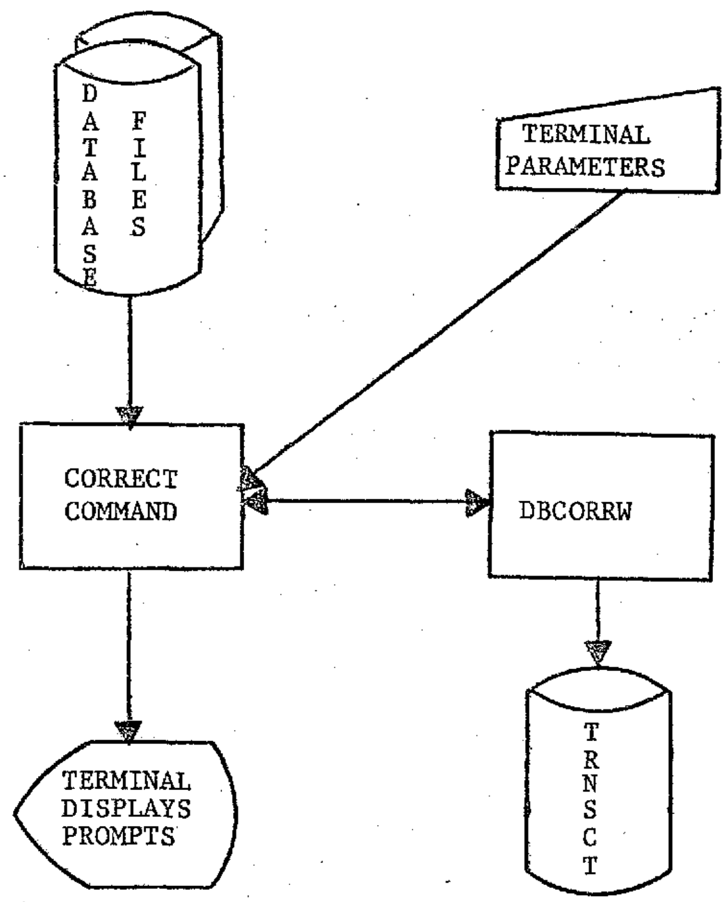


Figure 1. I/O Block Diagram

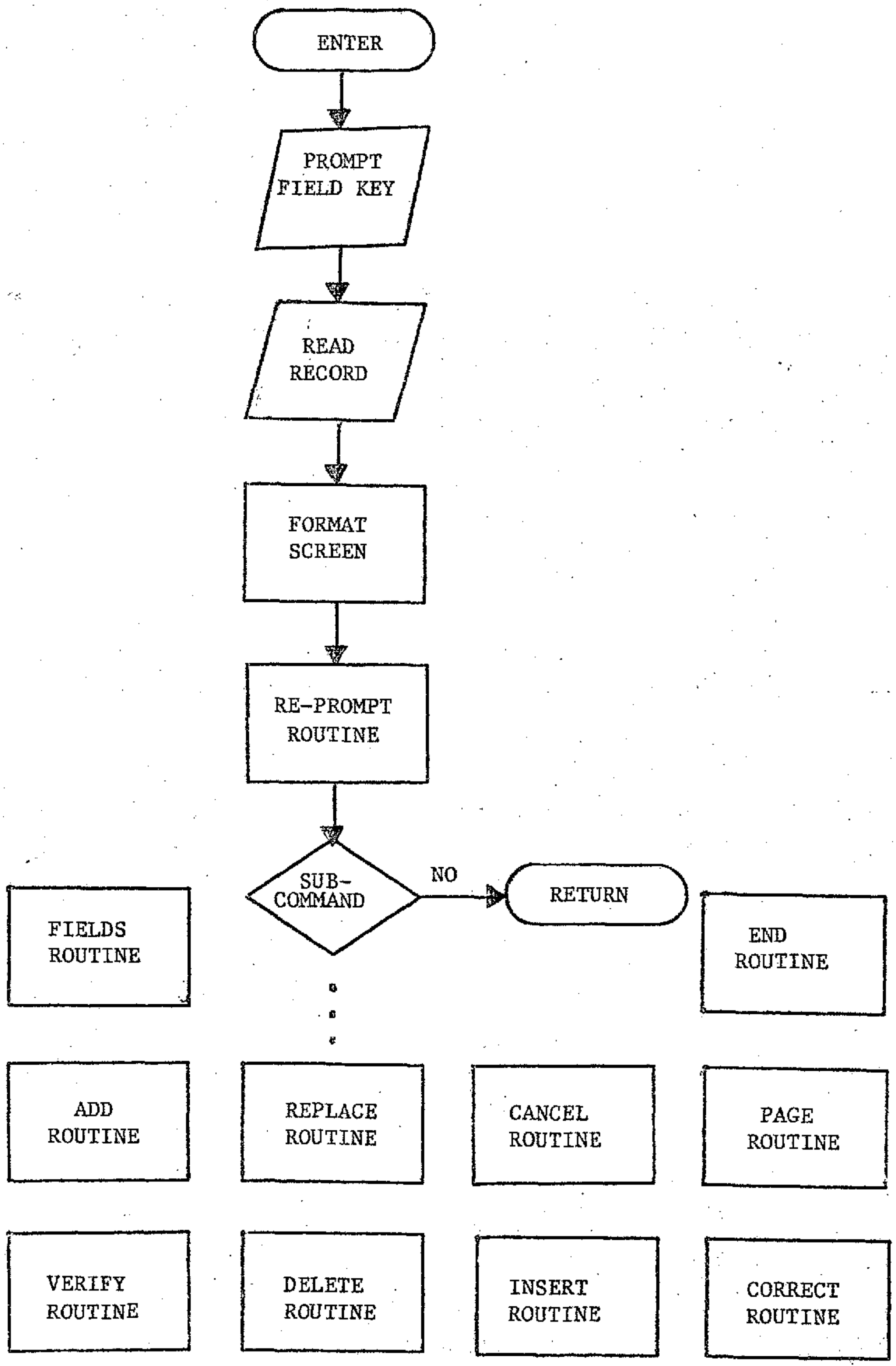


Figure 2. Top Level Flowchart

TOPIC F.16 - CORRECT COMMAND - WRITE TRANSACTIONS

A. MODULE NAME

Program-ID - NDBCORRW
Module-ID - DBCORRW

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The DBCORRW module is the routine that writes the transactions for the CORRECT command to the TRNSCT file. It is called by the DBCORR module.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The only output file from the DBCORRW module is the transaction file. This file is a QISAM data set containing maintenance transactions from all sources for all files. The fields of the transaction file and their format are completely described in the

Dataset specifications.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROGRAM REQUIREMENTS

1. Flowchart

See Figure 2

2. Narrative

a. Output Routine

This routine analyses the data maintained for the field being processed, and for each element whose data has been changed, creates transactions to represent the change. The routine calls write tranplx to actually write the transactions. The routine handles three cases, an added element, a deleted element, and a changed element. Upon completion, the routine returnes to its caller.

b. Write Tranplx

This routine performs the actual creation of transactions, based upon the data supplied to it. Upon completion, it returns to its caller.

F. CODING SPECIFICATIONS

1. Source Language

The NDBCORRW orogram employs the IBM PL/I programming language. The special extension of that language, called TSPL/I is utilized for all terminal communications.

2. Suggestions and Techniques.

Not Applicable

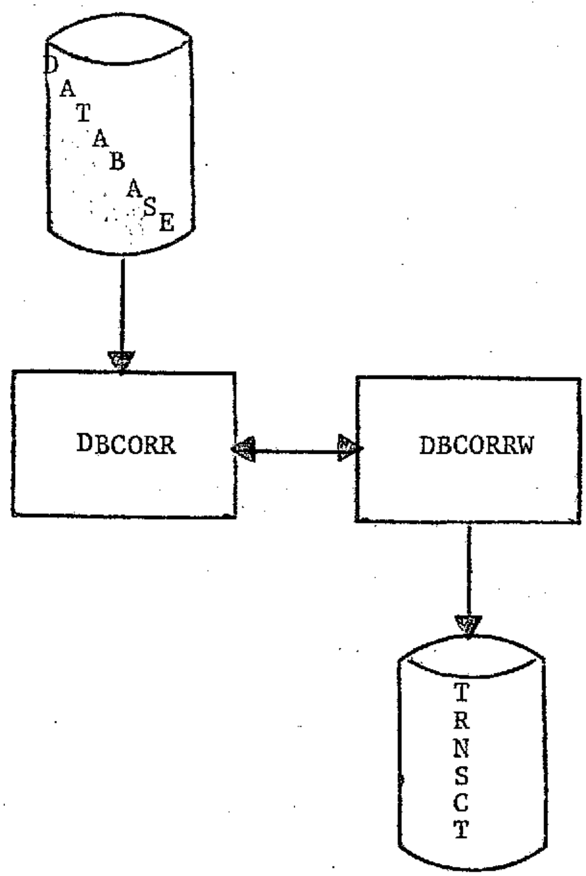


Figure 1.   I/O Block Diagram

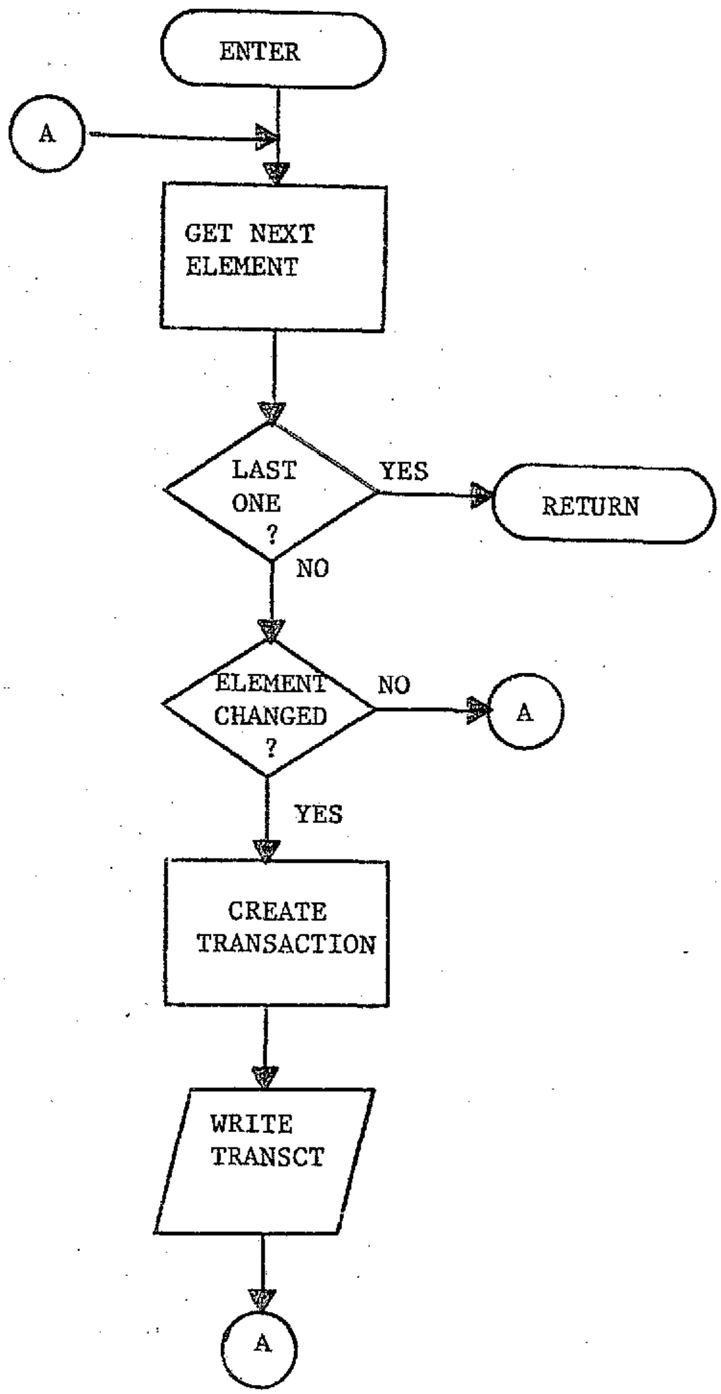


Figure 2. Program Flowchart

TOPIC F.17 RETRIEVAL DISPLAY COMMAND

A. MODULE NAME

Retrieval, DISPLAY Command (module 2 of 2)
Program-ID - NDBDSPLA
Module-ID - DBDSPLA

B. ANALYSTS

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The DISPLAY command allows the retrieval system user to have designated data for a given set to be displayed on a terminal. Like the PRINT command, the user may specify the format of the output as the citation number, the citation, the abstract, or the full text for any item contained in a set which has been previously selected. Optionally, the user may prespecify a format of his own, using the FORMAT command, to govern the DISPLAY command. One set-number is reserved for special purposes in the system. Set-number 0 is a logical reference to the entire anchor file. The PAGE command also calls the DISPLAY command in order to create additional displays, logically, before and beyond the current one. The calling sequence is: DISPLAY set-number, format, item, type or, alternately, DISPLAY citation#, format.

This module is called by DBDSPL via DBINIT to build the screen images for the DISPLAY requests.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

The anchor and associated files of a dataplex will be input to the DISPLAY command. The complete description of the files in a dataplex is found in the Data Set Specification Section of the Workbook.

d. On-line Terminal Entries

A terminal is the most likely source of the parameters which are passed to the DISPLAY command. The parameters available to the DISPLAY command are set or citation number, format, items, and type. The NASIS system will apply default values to the parameters, if they are available, when no original values are entered.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

The DISPLAY command will output a partially-formatted display of the items in a set or for a specific citation number. For sequential formats, each field is started on a new line, and the key field is always on the first line below the header information for a particular display. For columnar formats, the fields from each record are arranged across one or more lines in columns. The content of the display depends upon the format code entered as the second parameter.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COLFORM

The DISPLAY command refers to a COLFORM table when a columnar format is referenced.

b. USERTAB

This table contains user-oriented and status information.

c. FLDTAB

The DISPLAY command refers to FLDTAB to locate the appropiate sequential (SEQFORM) or columnar (COLFORM) format table.

d. RETDATA

This table contains data fields unique to the retrieval sub-system.

e. PLEX

The DISPLAY command uses a DBPL/I file called PLEX for all of its retrievals from the dataplex.

f. SEQFORM

The DISPLAY command refers to a SEQFORM table when a sequential format is referenced.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBDSPLA

This module is called by the DBDSPL module via the DBINIT Transient Module Interface convention; DBDSPLA uses necessary information from the common structure DSPLCTL.

b. Build Screen Image

This is a common routine for building a DISPLAY screen image either for an original DISPLAY command or for a PAGE command.

For a sequential format, field names are taken successively from the SEQFORM down to the number of field names.

In the most general case, each field consists of multiple elements and each element value is so long as to require multiple lines of a buffer. The first line for the first element of a field is tagged with the fieldname and a colon. The first line for an element after the first of a field is tagged with only the colon. Successive lines after the first for an element have their tag entirely suppressed. The degenerate cases of a single element field and/or an element short enough to fit on one line of the buffer are handled. And if the field is null (no data present), nothing is posted to the buffer at all for that field name.

Subfile resident fields are displayed similar to multiple elements; however, the first element of the field per subfile record has the field name tag duplicated, and a special heading is displayed (depending on the "type" parameter) as each new subfile record is processed.

If the field names are not all processed before the bottom line of the buffer is reached, the routine is left in such a state that it will resume where it left off if normal forward paging is attempted. But if the field names are all used, then the remaining lines are cleared.

For a columnar format, the optional page number, title, and header lines are copied into the buffer. Then field names are taken successively from the COLFORM, and used to retrieve the field values which are arranged across a line of the buffer. If there are any multiple element fields, futher lines of the buffer are used for remaining elements until the record's desired fields have all been retrieved. If there are any further records in the set, the next record is read and the process repeated. When the buffer is full, the routine is left in such a state that it will resume where it left off if normal or skip paging is attempted. But if the data is exhausted, then the remaining lines are cleared.

c. Write Screen

Using the full screen mode of output, the

current screen image is displayed on the terminal.

d. Return

Do a normal return to the calling routine.

3. Submodules Required

a. DB - data base package

b. PSTRAT - save strategy

c. TS - terminal support package

d. DBSETU - set information package

e. DBFLDU - field utilities

F. CODING SPECIFICATIONS

1. Source Language

The DISPLAY command is coded entirely with the IBM PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the dataplex. The TSPL/I language extension handles all instances of communication with the terminal.

2. Suggestions and Techniques

Not Applicable

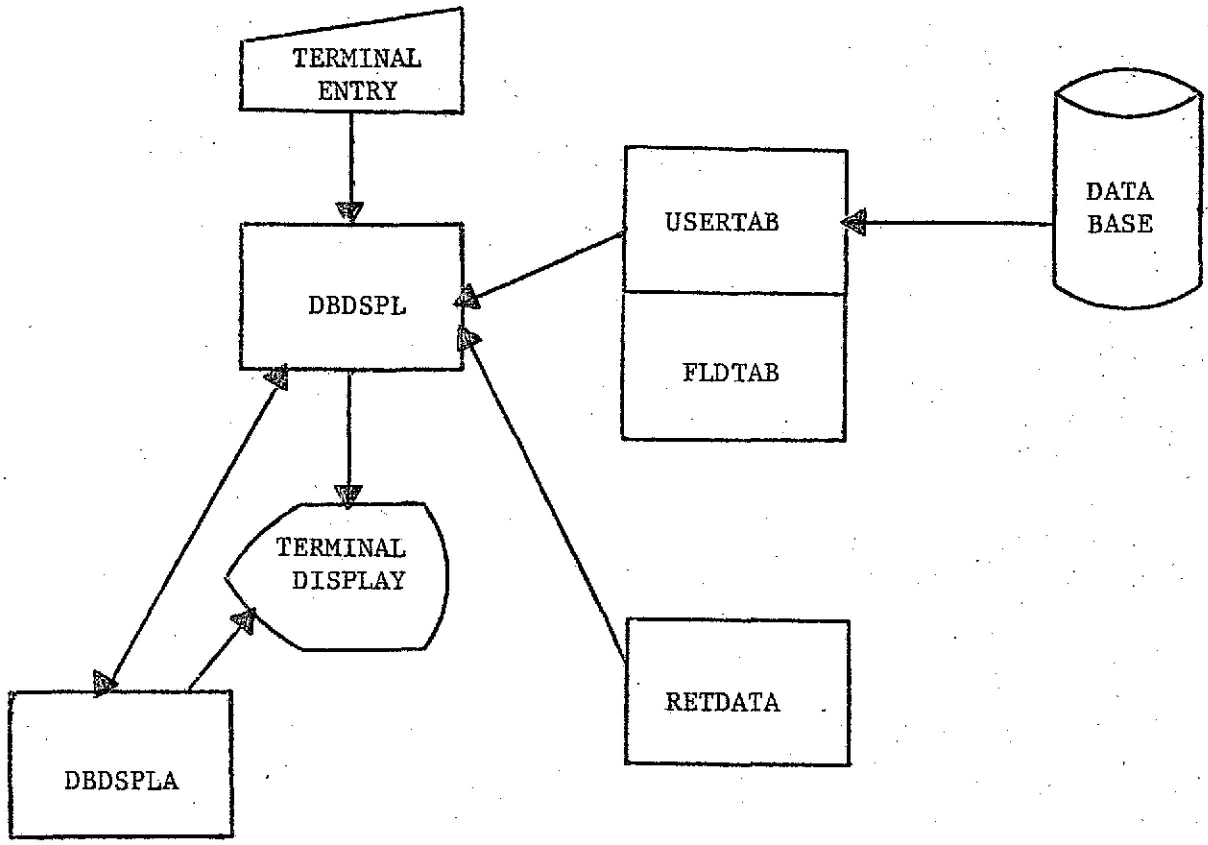


Figure 1. I/O Block Diagram

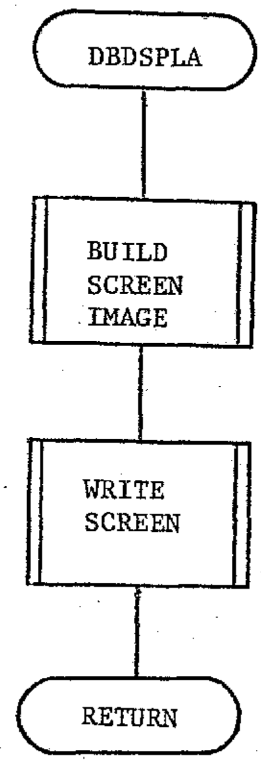


Figure 2. Top Level Flowchart

TOPIC F.18 - RETRIEVAL FORMAT COMMAND

A. MODULE NAME

Retrieval, FORMAT Command (module 2 of 2)
Program-ID - NDBFORMA
Module-ID - DBFORMA.

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

The DBFORM module is the first FORMAT command routine, called by the retrieval subsystem, whose purpose is to allow the retrieval system user to define, revise and/or display the content and format for subsequent information retrievals using the DISPLAY or PRINT retrieval commands. Sequential and columnar formats may be defined.

The DBFORMA module is called by DBFORM module to handle the processing of FIELD and HEADER subcommands. Refer to DBFORM Program Design Specification for further details.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

A terminal is the most likely source of the parameters which are passed to the FORMAT

command by the Terminal Support system. The fundamental parameters are the format number and the field names. Default values for the fundamental parameters are unlikely. The FORMAT command then accepts the FORMAT subcommands and their parameters.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COL_FORM

When the FORMAT command processes a new columnar format, it allocates and initializes a COL_FORM structure and posts its base address in the COL_FORMAT array in FLDTAB. When the FORMAT command processes a TITLE or HEADER sub-command or any other revision to a columnar format, it updates the appropriate COL_FORM structure. Thus, a COL_FORM structure specifies a columnar format for use by the DISPLAY and PRINT commands.

b. FLDTAB

The FORMAT command refers to the DATAPLEX portion of FLDTAB. The FORMAT command also posts the SEQ_FORMAT and COL_FORMAT arrays as it processes new formats.

c. SEQ_FORM

When the FORMAT command processes a new sequential format, it allocates and initializes a SEQ_FORM structure and posts its base address and field name count in the

SEQ_FORMAT array in FLDTAB. Thus, a SEQ_FORM structure specifies a sequential format for use by the DISPLAY and PRINT commands.

d. USERTAB

The FORMAT command checks the USERTAB.RETRIEVE switch to verify that it is being called properly.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2 of DBFORM Design Specification.

2. Narrative

a. Format

The FORMAT command is recognized by the retrieval system director module DBINIT which calls the DBFORM entry point. When a FIELD or HEADER subcommand is recognized, the DBFORMAmodule is called via the DBINIT Transient Module Interface convention. These subcommands are described below.

b. Process FIELD command

FLDSPEC parameter values are obtained one by one from the Terminal Support system and processed individually. The field names are checked for existence in the current database by a call to the field utilities (DBFLDU). If a field name or position is invalid, a diagnostic is issued and the keyboard unlocked for re-entry of that field name with any options or default for that field to be ignored. Normally, for sueqential formats, the field name is posted in SEQ_FORM, or for columnar formats the field name, position (proportioned, if not specified by the user) and options are posted or updated in COL_FORM.

c. Process HEADER command

If the current format is not columnar, the HEADER command is cancelled with a diagnostic message.

A HDRLINE parameter value is obtained from

the Terminal Support system, if the user entered it explicitly, or by assuming the next relative header line number. The value is checked for syntax, range, duplication, and space in COL_FORM.TOP. Any error is diagnosed and the user allowed to re-enter the parameter. For a header line deletion, any lower header line images are shifted up and COL_FORM.TOP#HEADERS is decremented and control branches to (c.). For a new header line, any lower header line images are shifted down and intervening lines blanked in COL_FORM.TOP.LINE and COL_FORM.TOP.#HEADERS is posted. Thus a current header line is determined for the following processing.

If no HDRSPEC parameter values were entered explicitly by the user, every column accross the current header line has its field name value centered over it and control branches to (c.).

Otherwise, HDRSPEC parameter values are obtained one by one from the Terminal Support system and processed individually. If only a literal value is given, it is centered over the next column to the right. If only a parenthesized field name is given, it is centered over the column for the field name. If both a literal value and a parenthesized field name are given, the value is centered over the column for the specified field name. Any syntax, field name, or past rightmost column error results in a diagnostic message allowing the user to re-enter one value or to default for that value to be ignored.

3. Submodules required

DBFLDU - field utilities
DBUCHEK - check name routine
GETSFMT - get stored format
PUTSFMT - put stored format
PSTRAT - save strategy
TS - terminal support package

F. CODING SPECIFICATIONS

1. Source Language

The FORMAT command is coded in IBM PL/I. The TSPL/I language extension is used for all

communication with the terminal.

2. Suggestions and Techniques

The PSTRAT external routine shall be called whenever a valid command or subcommand with valid parameters is detected.

Subroutine facilities shall be coded to handle the general case of re-proportioning columns and re-centering headers. (DUP_COL, RE_PRO_COL, RE_HEAD).

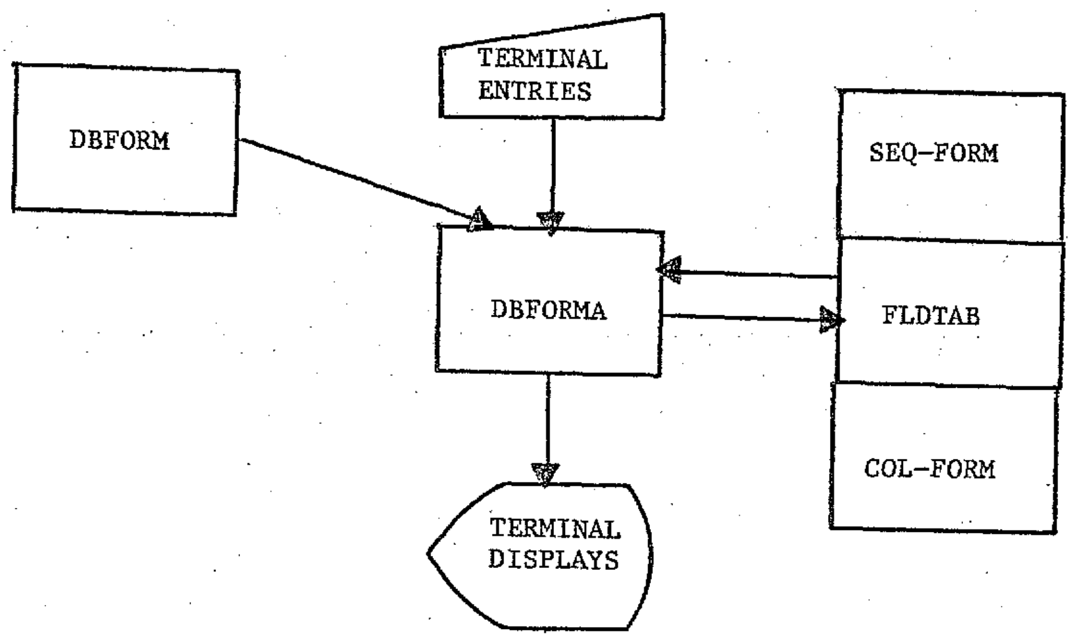


Figure 1. I/O Block Diagram

TOPIC G.1 - ACCUMULATION

A. MODULE NAME

Statistics Accumulator
Program-ID - NDBACCUM
Module-ID - DBACCUM

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

Primarily, this module is used to accumulate the maintenance statistics on those data bases which have already been loaded. This is part of the initialization process for the usage statistics function.

This program reads an existing data base anchor file and accumulates the number of records on it. Then, it posts this record count to the STATIC file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      The data base which is to have the statistics accumulated, and the STATIC file.

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

a. Output Files

The STATIC File

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

This program is relatively simple and the execution time should be small. Therefore, any serious errors will output a simple message and abend the job.

The message is as follows:

ERROR ON $01 OF $02.

Where:
$01 is the ONFILE,
$02 is the ONCODE.

The program will accept the data base name as a parameter and will proceed to count the anchor files records. When this task is completed, it will open the STATIC file for update and post the record count.

The posting of the STATIC data base assumes that no record for this data base currently exists. Therefore, if an error occurs on the LOCATE Statement for the posting, the new record is posted over the existing one if that is the error, otherwise the job is terminated. The key's value for the locate statement is as follows:

A value of '0' concatenated to the data base name and filled with $'s to 32 characters.

The field 'ANCOUNT' is posted with the number of anchor records.

The field 'MAINDATE(1)' is posted with the jobs run date, i.e., this is assumed to be the creation date for statistics.

The field 'TOTAL RUN' is posted with a '1'.

The field 'TRANCNEW' is posted with the number of anchor records.

The following fields are posted to '0': 'TOTALTRN', 'TRANCDEL', 'TRANCUPD', 'TRINVNEW', 'TRINVDEL', 'TRINVUPD', 'TRSUBADD', 'TRSUBDEL', and 'TRSUBUPD'.

F. CODING SPEICIFCATIONS

1. Source Language

As much as possible of the DBACCUM module is coded in the IBM programming language PL/I. The input and output coding for access to files in a data base is handled through an extension to that language known as DBPL/I.

2. Suggestions and Techniques

It is important to remember that the executive error '99' indicates an end of file condition. Special attention is made for the handling of the data base executive errors.

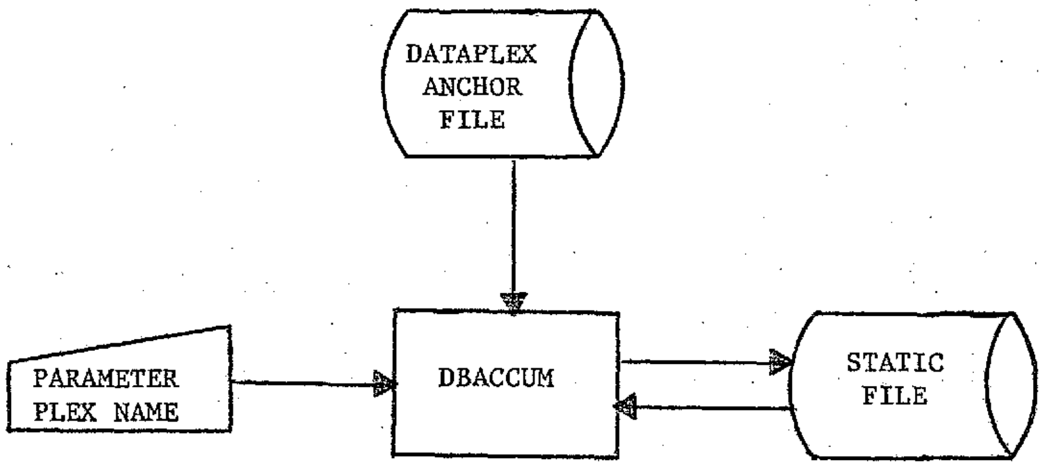


Figure 1. I/O Block Diagram

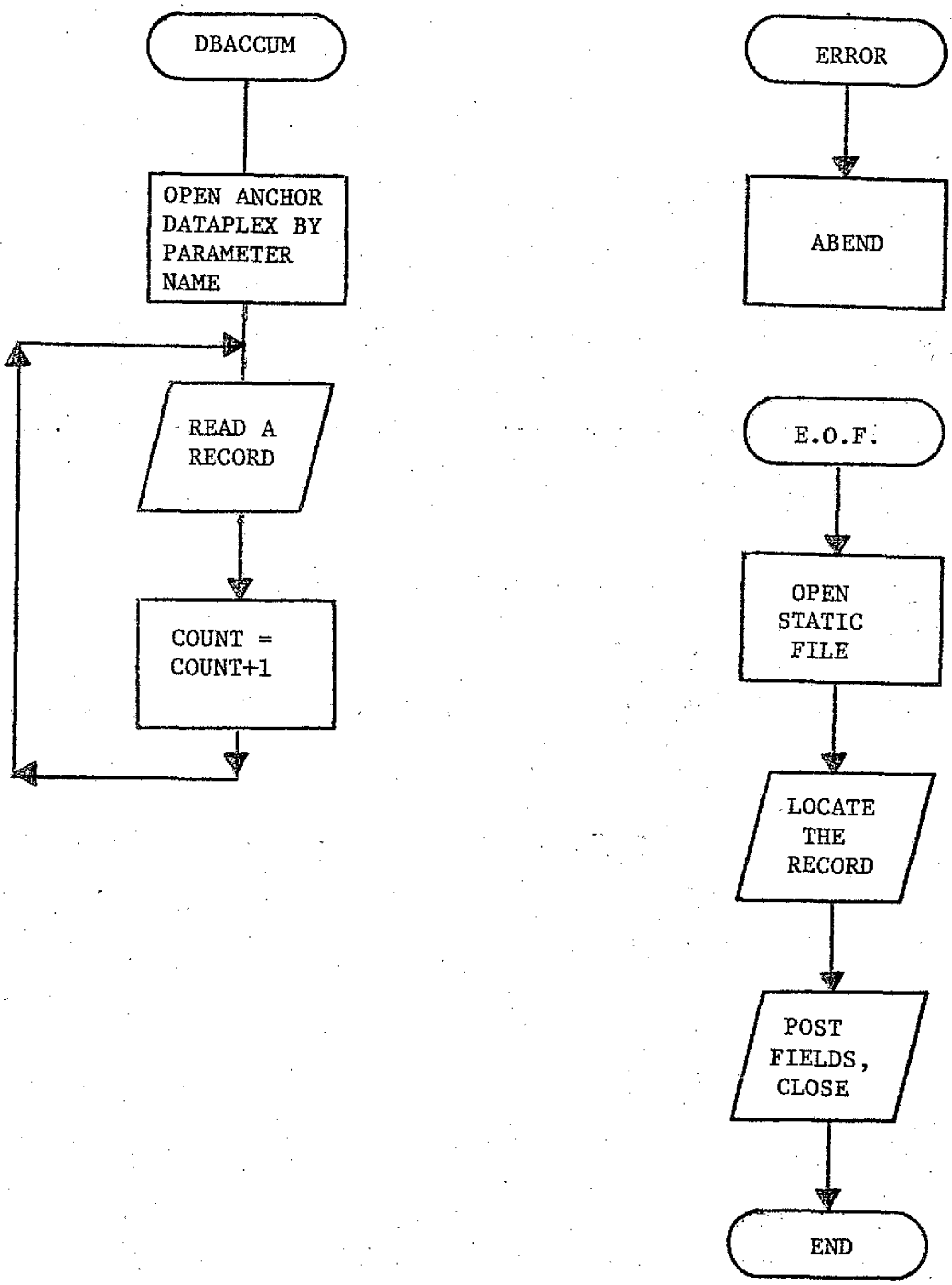


Figure 2. Top Level Flowchart

TOPIC G.2 - REPORT PRINT

A. MODULE NAME

Print the Retrieval Statistics
Program-ID - NDBPRNTR
Module-ID - DBPRNTR

B. ANALYST

Edward J. Scheboth, Jr.
James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

The purpose of this program is to present a detailed listing of the contents of the STATIC file pertaining to retrieval statistics. Summaries of various germane items are made as the module develops the required report.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The STATIC file, (for full details on this file see Section III of the Development Workbook).

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

The retrieval statistics' report, (for full details of this report (listing) see Section III of the Development Workbook).

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Flowchart

See Figure 2

2. Narrative

This module performs the following logic in order to produce the retrieval statistics' report

a. Open the STATIC file for sequential input (use DBPL/I).

b. Read the STATIC file sequentially record by record and while reading, construct from the current information on the STATIC file the required listing.

c. Output the print file required to produce the retrieval statistics' report.

d. Close all files: Terminate.

Note: It will be necessary for this program to accumulate various information so that it can output the summary of retrieval statistics, representing all of the statistics on the STATIC file.

F. CODING SPECIFICATIONS

1. Source Language

As much as possible of the DBPRNTR module is coded in the IBM programming language PL/I. The input

and output coding for access to files in a data base is handled through an extension to that language known as DBPL/I.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications and all data base executive errors.

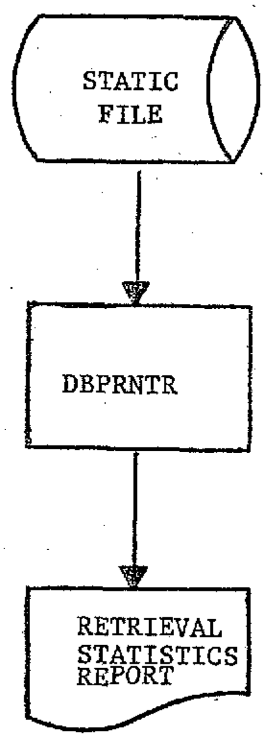


Figure 1. I/O Block Diagram

TOPIC G.3 - USAGE STATISTICS UPDATE

A. MODULE NAME

Update Maintenance Statistics
Program-ID - NDBUPDST
Module-ID - DBUPDST

B. ANALYST

Edward J. Scheboth, Jr.
James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

This program updates the statistics file (STATIC) with the maintenance statistics from the load/create program (DBLOAD) or from the maintenance mainline (DBMNTN).

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      The STATIC Dataplex

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      The STATIC File

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The parameters are passed via standard PL/I procedure/procedure linkage key calls from DBMNTN and DBLOAD.

The parameters which are passed are as follows:

a. Calling program identifier character 2.
   First Character
      C = first call.
      M = subsequent call.
   Second character.
      L = called from LOAD.
      anything else signifies - called from elsewhere.

b. File being updated.

c. Number of new anchor records, character 6.

d. Number of deleted anchor records, character 6.

e. Number of updated anchor records, character 6.

f. Number of new subfield records, character 6.

g. Number of deleted subfile records, character 6.

h. Number of updated subfile records, character 6.

i. Number of new inverted records, character 6.

j. Number of deleted inverted records, character 6.

k. Number of updated inverted records, character 6.

The load/create module (DBLOAD) invokes this module only once, and this is at the end of the create run. Therefore, this module opens the STATIC file for direct (update or output) and locates the new record. The data is put and the file closed.

The final call from the maintenance mainline will have an 'F' posted to the calling program identifier.

If the updating of the STATIC file is successful, a 'G' is posted to the calling program identifier upon return; whereas, if the results are not successful, a 'B' is posted.

If the results of the attempted posting are bad, the calling programs will resolve the disposition of the non-posted data.

The details of the contents of the STATIC file can be found in Section III of the Development Workbook.

The following illustrates the parameters passed and the associated fields which are updated; they are in the form "parameter - static field name":

a. Maintenance date - MAINDATE

b. Number of new anchor records - TRANCNEW

c. Number of deleted - TRANCDEL

d. Number of update - TRANCUPD

e. Number of new subfile records - TRSUBNEW

f. Number of deleted - TRSUBDEL

g. Number of updated - TRSUBUPD

h. Number of new inverted records - TRINVNEW

i. Number of deleted - TRINVDEL

j. Number of update - TRINVUPD

k. Calling program identifier - *-none-*

It is important to remember that there is a one for one correspondence between all of the previously mentioned STATIC file fields. For Example:

If MAINDATE = '03/16/70' and this is the actual date of the maintenance run, then if the MAINDATE value of '03/16/70' is the third element in the variable length field, then all updates to the other elemental fields of the record are made to the third element.

The table which follows will help to illustrate this more clearly.

MAINDATE	01/16/70	02/16/70	03/16/70	null
TRANCNEW	9	3	1	
TRANCDEL	18	4	1	
TRANCUPD	3	7	1	
TRSUBNEW	7	9	1	
TRSUBDEL	3	12	6	
TRSUBUPD	1	9	2	
TRINVNEW	16	3	1	
TRINVDEL	4	4	1	
TRINVUPD	12	7	1	

The fields we are concerned with are: MAINDATE, TRANCNEW, TRANCDEL, TRSUBNEW, TRSUBDEL, TRSUBUPD, TRANCUPD, TRINVNEW, TRINVDEL, TRINVUPD.

These fields are all variable length fields with multiple fixed length elements, treated as 13 element arrays. The first element in the array is used as an accumulator. Elements 2 through 13 are used to represent individual maintenance runs.

This is simple enough--when this module is called from DBMNTN, it simply locates the maindate which is the same as the parameter and does the posting into that given element.

The question is what does this module do when it's called for the first time from the maintenance program (DBMNTN) and the date is not equal to any of the posted dates and all 13 elements have data so that there is no additional elemental slot where the data can be placed.

The solution is as follows: First, the second elemental slot is 'REPUT' to null. Which causes the file executive to automatically slide all of the other elements (logically). Then, the new maintenance data will be 'PUT' as the thirteenth element.

F. CODING SPECIFICATIONS

1. Source Language

As much as possible of the DBUPDST module is coded in the IBM programming language PL/I. The input and output coding for access to files in a file is handled through an extension to that language known as DBPL/I. All terminal communication is handled through the terminal support preprocessor TSPL/I.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications and all file executive errors.

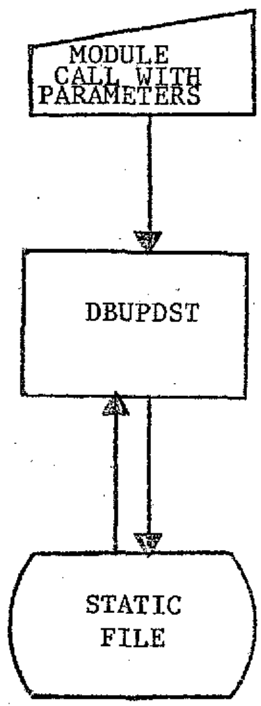


Figure 1. I/O Block Diagram

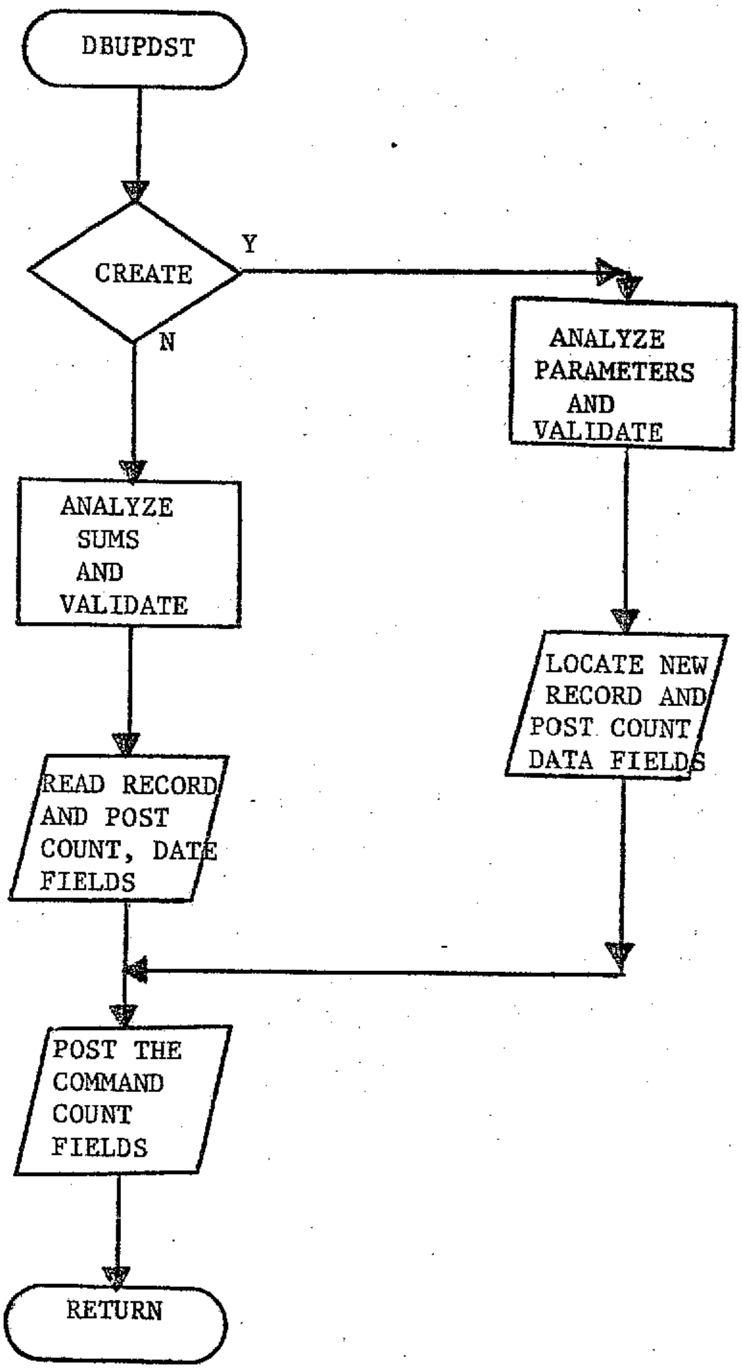


Figure 2. Top Level Flowchart

TOPIC G.4 - CLOCK ROUTINES

A. MODULE NAME

Clock Routines
Program-ID - NTIMERS
Module-ID - TIMERS

B. ANALYST

Edward J. Scheboth, Jr.
Neoterics, Inc.

C. MODULE FUNCTION

This module initializes two internal clocks, one for CPU time and the other for CONNECT time. These clocks may be read at a later time to provide the elapsed time plus initial values.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

In the START entry, the initial values are assigned to the total clock value and an even odd pair of clocks are started even (0) with task option ODD(1) with real option and two counters are set with these values.

In the READ entry, a flag is set to on at entry. The clocks are read and the initialized totals are updated. The clocks are stopped and restarted to prevent expiration, the values are provided to caller the 0-1 pair of clocks started, the indicator turned off and return made to caller.

In the STOP entry, the two counters of clock numbers are deducted by 2 and each pair of active clocks stopped.

If either clock should expire, the expiration routing post full values to total and starts a new clock with value +2 and returns.

F. CODING SPECIFICATIONS

1. Source Language

   Assembler

2. Suggestions and Techniques

   Not Applicable

TOPIC G.5 - STATIC REPORT

A. MODULE NAME

Maintenance Statistics' Report
Program-ID - NDBPRNTM
Module-ID - DBPRNTM

B. ANALYST

Edward J. Scheboth, Jr.
James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

This program opens and reads the STATIC file (sequential input); analyzing, accumulating and formatting (for printing) the maintenance statistics' information which is currently posted. The end result is a maintenance statistics' report. It has the added function of snapshot dump and re-initializing the seven variable element fields which are the running totals of the maintenance statistics.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The STATIC file (for detailed and complete information on this file refer to Section III of the Development Workbook).

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      Not Applicable

   c. Formatted Print-outs

      The maintenance statistics report (for complete detailed information on this listing refer to Section III of the Development Workbook).

4. Reference Tables

   Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   This module performs the following logic in order to produce the maintenance statistics' report:

   a. Opens the STATIC file for sequential input (use DBPL/I).

   b. Read the STATIC file sequentially, record by record, and while reading constructs from the current information on the STATIC file, the required listing.

   c. Outputs the print file required to produce the maintenance statistics' report.

   d. Snapshots the ten variable element fields if they are full.

   e. Close All Files: Terminate.

   Note: It is necessary for this program to accumulate various information so that it can output the summary of maintenance statistics.

F. CODING SPECIFICATIONS

1. Source Language

As much as possible of the DBPENTM module is coded in the IBM programming language PL/I. The input and output coding for access to files in a file is handled through an extension to that language known as DBPL/I. All terminal communication is handled through the terminal support preprocessor TSPL/I.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications and all file executive errors.

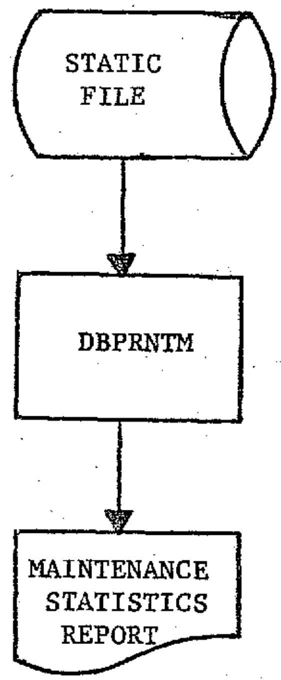


Figure 1. I/O Block Diagram

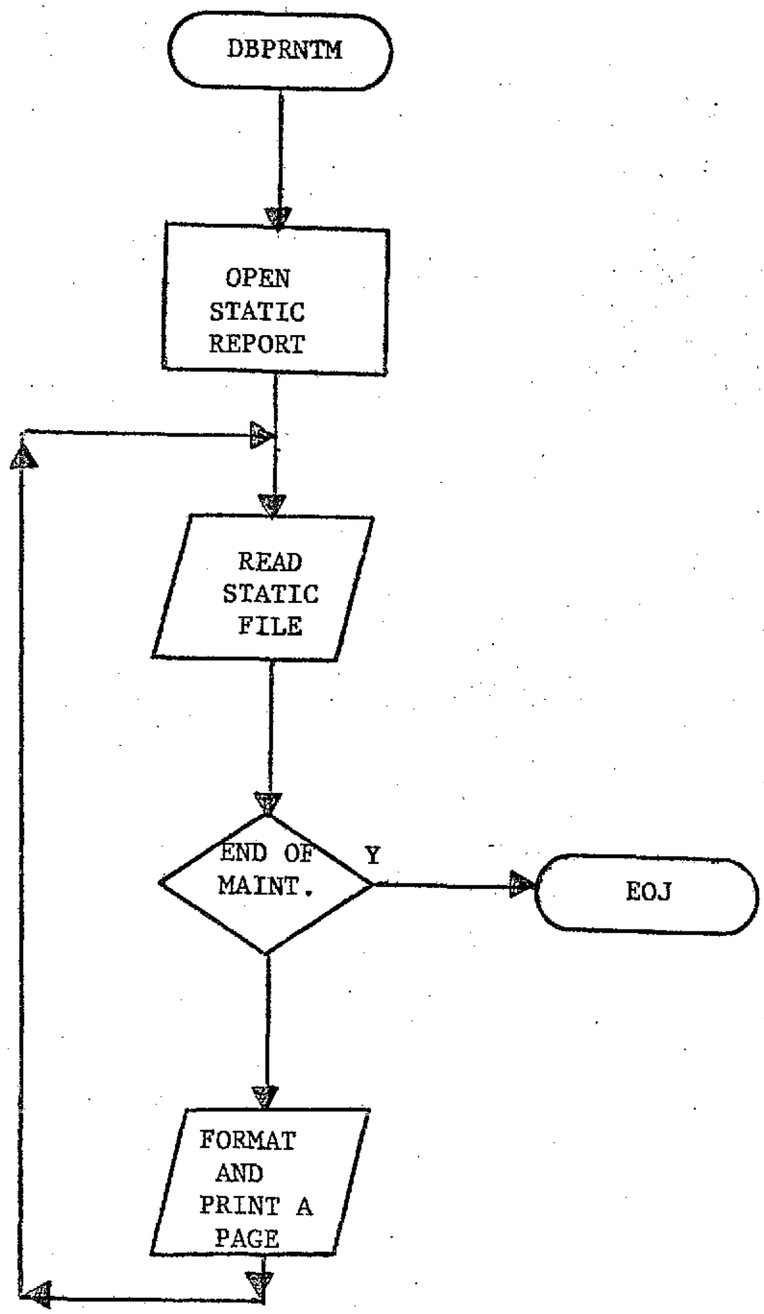


Figure 2. Top Level Flowchart

TOPIC G.6 - RETRIEVAL STATISTICS DIRECTOR

A. MODULE NAME

Retrieval Statistics Director
Program-ID - NDBSTAT
Module-ID - DBSTAT

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

This module is the heart of the retrieval statistics. It has an entry point for each retrieval module included in the statistics.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      The Static Dataplex.

   b. On-line Terminal Displays

      Not Applicable

c. Formatted Print-outs

See CHKPT dump

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The MFCE is used to convert inverted indices to data base file names.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Mainline:

The INIT entry checks to see if there was a crash during the last session by the existence of the ONES record and then write one if there wasn't one or after Checkpoint Routine deletes it. INIT initializes statistics that like INIT in the command system setting up the necessary tables or pointers for later use.

Each command entry, one each for EXPAND, SELECT, SEARCH and CORRECT, pushes its information, command type, NASISID OWNERID and fill, into the stack and then checks to see if it is time to update the statistics by checking the command count and entry count for critical level.

The DBSTATF entry call on termination of a session, just indicates that this is to be the end and provides strategy information and branches to the PUTSTAT routine.

The DBSTATD entry deletes this strategy from the statistics if it is there.

The PUTSTAT routine always updates the CPU and connect time by calling the TIMERS routines for their values. It also always pops the command stack and updates each command count and the set-date for the specified file. The stack is a FIFO stack, a one dimensional structured array. If this is a DBSTATF entry, then the strategy

information 'STRATSTR,' and 'STRATLEN' and usage information 'LASTDATE' are complete updated. Finally, for the DBSTATF entry to update the storage allocation is freed and the ones record deleted from STATIC.

Checkpoint:

The initialization entry point of DBSTAT enters the checkpoint routine when it detects the presence of a record in the STATIC file with a key that has all 'bits on'. This condition indicates that the system crashed, during the last NASIS command session, before job completion. The initialization module writes the record with the key of all 'bits on' (x'FF') to the STATIC file, either after return from check pointing or upon detection of no record.

The initialization module enters the checkpoint routine after a normal terminal session has been completed.

The NDBCHKPT programs operation is relatively simple. It will read sequentially from the beginning to the end of the STATIC file and perform the checkpoint function. Upon completion, it will delete the record that had all 'bits on' for the key's value.

It should be mentioned at this point that the data base executive returns a '99' value as an error code when the end of data is encountered. For the other data base executive errors, please refer to the data set specs.

If an error is detected while trying to open the STATIC file for update (direct), this program will abend. It will be attempted automatically at the beginning of the next terminal session.

The checkpoint function itself is relatively simple. It consists of looking at the record, determining if it needs to be re-initialized and then either re-initializing and printing it or getting the next record. The details of this process are as follows:

> There are five fields whose field names are #EXPANDS, #SELECTS, #SEARCHS, #CORRECTS, and SESSDATE.

These fields are treated as 13 element arrays.

When there is data in existence in all 13 of these elements, it is time to re-initialize the record so that the data from all subsequent NASIS command sessions can be posted.

The first element of these fields represents an accumulator. The second through the thirteenth elements represent the data from individual NASIS command sessions.

When the second through the thirteenth elements have actual data accumulated in them, then, this program will add each of the elements (2-13) common to a particular field into the first element. The second through thirteenth elements will then be 'nulled'. The snapshot will be printed for this record upon detection of all 13 elements having data and before the re-initialization.

The next NASIS command session will begin accumulating data in the second element. The first element will NEVER be deleted.

F. CODING SPECIFICATIONS

1. Source Language

PL/I and DBPL/I

2. Suggestions and Techniques

Not Applicable

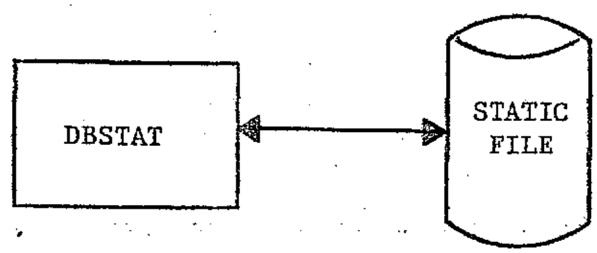


Figure 1. I/O Block Diagram

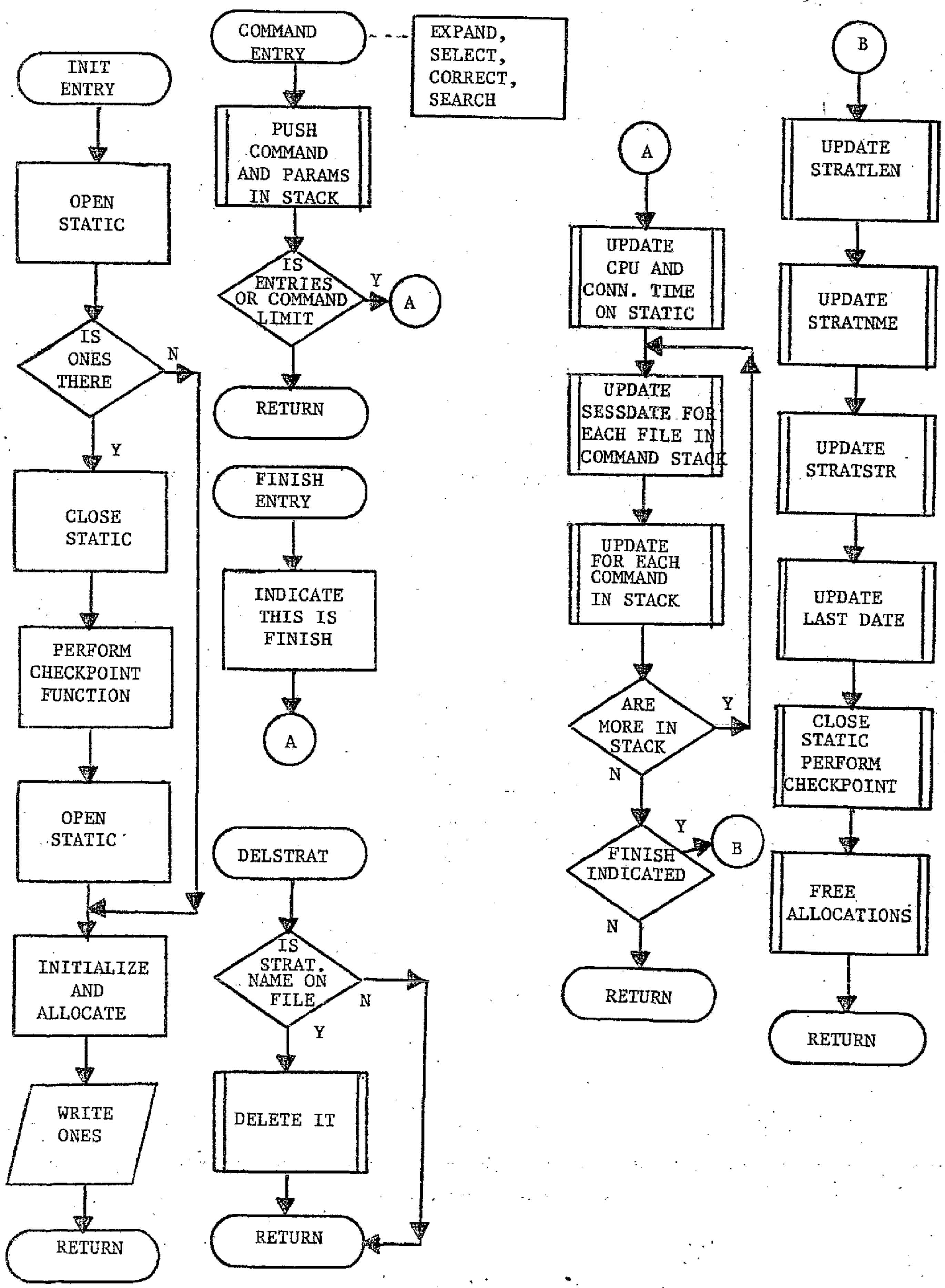


Figure 2. Top Level Flowchart

TOPIC H.1 - EXPLAIN FACILITY

A. MODULE NAME

Program-ID - NDBEXPL
Module-ID - DBEXPL

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module allows the user to display the explanation of a message or term, the origin of a message or the responses to a prompt, that has appeared on the screen, or, the text of any of the standard prompting messages on the message file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      This module receives its input in the form of parameters passed with the EXPLAIN or PROMPT commands.

3. Output Data Sets

   a. Output Files

      Not Applicable

b. On-line Terminal Displays

This module displays the requested information for the user on the terminal.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBEXPL1

Upon entry, the program analyzes the parameter to determine paging, PROMPT, or EXPLAIN processing. If paging, the processing continues at label DBEXPLP (see below). If PROMPT, processing continues at label DBEXPL2 (see below). Otherwise, the program initializes the variables that control execution and the displaying of data to the user. It also sets up the mechanism by which paging is to be accomplished.

Next the program prompts for the OPTION and MESSAGE parameters required for the EXPLAIN function. It verifies that the option selected is valid, and if so, branches to the appropriate routine.

For simple explains, i.e., message explanations, the OPTION is treated as an index, verified, and the line number set to a value of 100. If the OPTION is not a valid index, the request is treated as a term explanation. The OPTION is then treated as a qualified term and used to construct the message key which is used to locate the term's explanation. For response

explanations the live number is set to a value of 400.

In each of the above instances, control is passed to a routine which attempts to read a data record. If successful, the record is written to the screen and the process repeated, until the data has been exhausted, or the screen filled. At this time, the paging controls are set, the screen is displayed to the user and the program is terminated. If no data was found, the routine branches to an error routine which displays a message to the user and terminates the program.

If the original request was for a message origin, the OPTION is treated as an index, and if valid, the appropriate message key is obtained, displayed to the user, and the program is terminated.

b. DBEXPL2

At this label, the program initializes the varriables that control execution and prompts for the MESSAGE parameter. It then prompts for the INSERTS parameter list.

Once complete, the program attempts to display the message indicated with the specified inserts.

c. DBEXPLP

At this label the program re-initializes the variables that control execution and the displaying of data to the user. If the paging status data indicates that more data remains, the program uses this data to restore the proper program status and then branches to the routine which posts data to the screen. If no data remains to be displayed, the program simply terminates.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the IBM PL/I language.

2. Suggestions and Techniques.

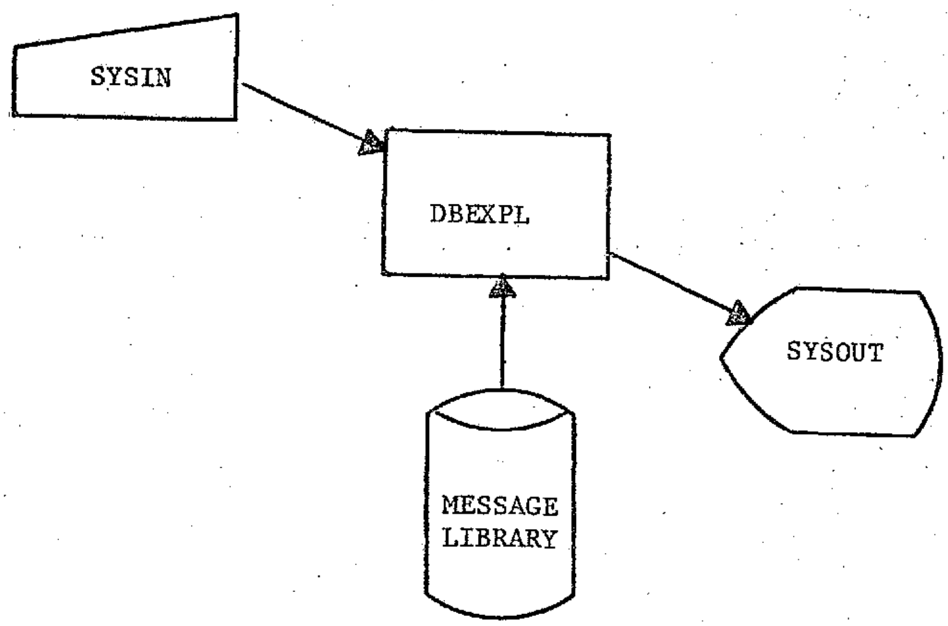


Figure 1. I/O Block Diagram

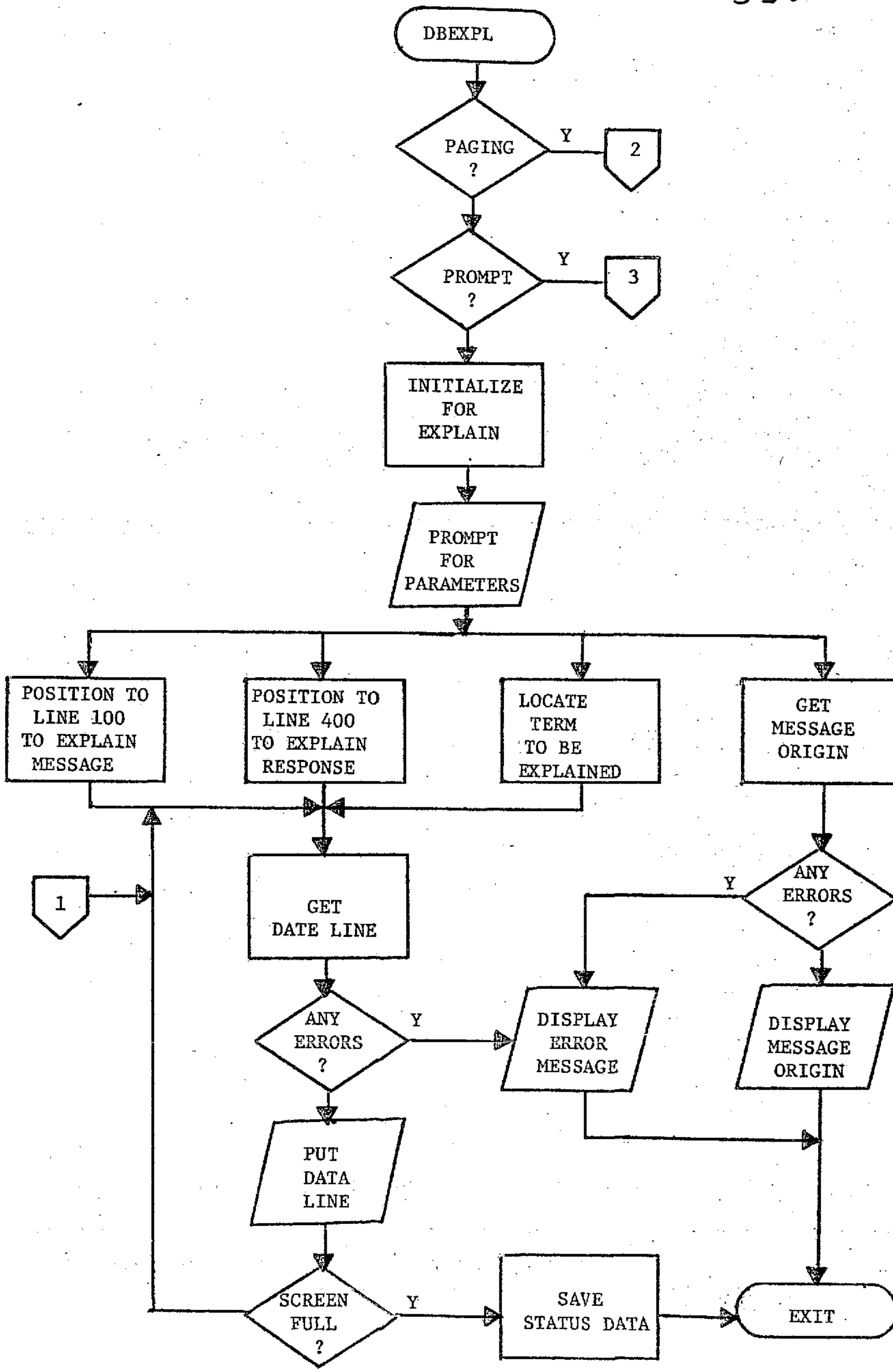


Figure 2A   Top Level Flowchart

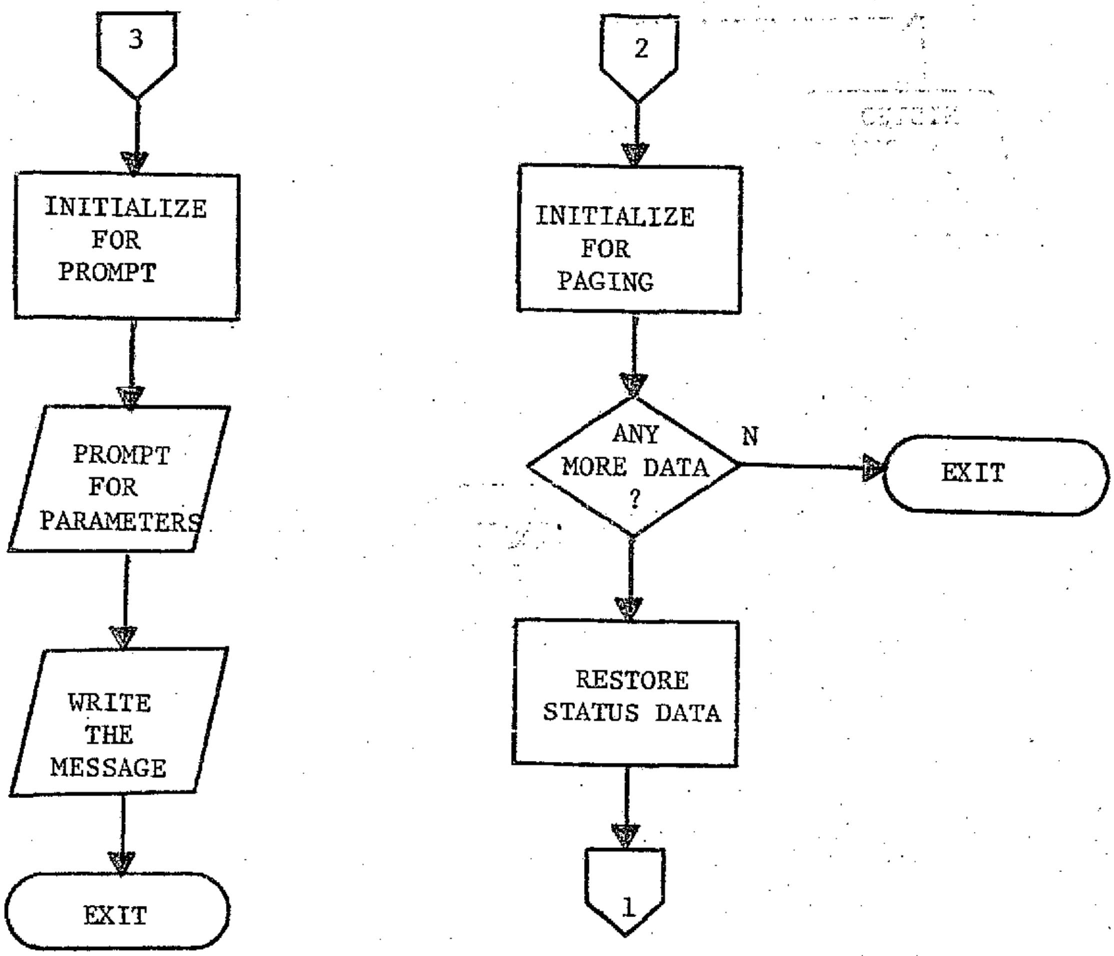


Figure 2B. Top Level Flowchart

TOPIC H.2 - STRATEGY INTERFACE

A. MODULE NAME

Program-ID - NDBSTRT
Module-ID - DESTRT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module serves as an interface between the strategy data set service routines and the rest of the NASIS system. In addition, it is the module which performs the functions specified by the FORMATS and STRATEGY commands, i.e., the listing of format and strategy names, the listing of strategies and the deletion of strategies.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      When serving as the processor for the FORMATS and STRATEGY commands, the program reads in the command and parameters specified by the user to invoke those commands.

3. Output Data Sets

   a. Output Files

Not Applicable

b. On-line Terminal Displays

When serving as the processor for the FORMATS and STRATEGY commands, the program produces the following formatted screen images,

1. Format names display

2. Strategy names display

3. Strategy display

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

USERTAB-is used to obtain the NASIS-id and to test the task status as represented by the various bit switches.

FLDTAB -is used to reference the formats currently defined for this user.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. GSTRAT

At this entry point the program initializes the parameter lists necessary to obtain a line from the strategy data set, and calls TSGETRG to do it. If an error occurs, and it is the first error for that region, a diagnostic message will be written to the user. Otherwise, the program simply returns to the caller.

b. PSTRAT

At this entry point the paragram initializes

the parameter lists necessary to write a line to the strategy data set, including the generation of the strategy or format name. It then calls ISPUTRG to perform the write. If PSTRAT is called and the TESTMODE, RERUN or RESTACT flags are set, the program immediately returns to the caller. If an error occurs while writing out the record, a diagnostic message is written to the user and the TESTMODE switch is turned on. The program then returns to the caller.

c. CSTRAT

At this entry point the program initializes the parameter lists necessary to change the name of a region. It then calls TSCHGRG to accomplish the change. If any errors are encountered, a diagnostic message is written to the user. The program then returns to the caller.

d. DSTRAT

At this entry point the program initializes the parameter lists necessary to delete a region of the strategy data set. It then calls TSDELRG to perform the deletion. If any errors are detected, a diagnostic message is written to the user. The program then returns to the caller.

e. DBSTRT1

At this entry point the program initializes itself to process the strategy command. It reads in the OPTION and STRATEGY parameters. The program then branches to the routine used to process the type of request specified by the OPTION parameter. If that parameter is not valid, the program writes a diagnostic message and terminates immediately.

If the user requested a strategy deletion, the program calls TSDELRG to delete the strategy specified. If an error occurs, a diagnostic message is written to the user. The program then checks for any additional names, and processes each in the same way. When all processing has been completed the program terminates.

If the user requested a listing of the

strategy names, the program initializes the screen and paging control data. It then repetitively calls TSGETSN to retrieve the names of the strategies. As each name is obtained, it is added to the output line and the line is written to the screen. When the screen is filled or when the strategies names are exhausted, the screen is displayed to the user, the paging status data is posted and the program is terminated.

If the user requested a listing of a particular strategy, the program initializes the screen and paging control data. The first strategy name specified is selected, and TSGETRG is repetitively called to obtain the lines comprising the strategy. Each line is posted to the screen. When the screen is filled or when the last line has been written, the screen is displayed to the user, the paging staters data is posted and the program is terminated. The paging status data must indicate when a strategy has been completely listed, so that the next name from the list can be used.

f. DBSTRT2

At this entry point the program initializes itself to display the names of the formats available to the user. It initializes the screen and the paging status data. The program then extracts the identifiers for all of the formats currently specified in the format tables. It then calls TSGETFN to retrieve the name of a stored format. It places the names of the formats on a line and writes the line out to the screen. The names are processed alphabetically, and as each stored format name is processed, a new one is read in. Stored formats that are also present in the format tables are only shown once. When the screen is filled, or when the list of names is exhausted, the screen is displayed to the user, the paging status data is posted and the program is terminated.

g. DBSTRTP

At this entry point the program re-initializes itself to the status saved before the last termination. If more data remains to be displayed, the program branches

to the proper routine to produce the next display screen. If nc more data remains, a diagnostic message is written to the user and the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the IBM PL/I language.

2. Suggestions and Techniques

Not Applicable

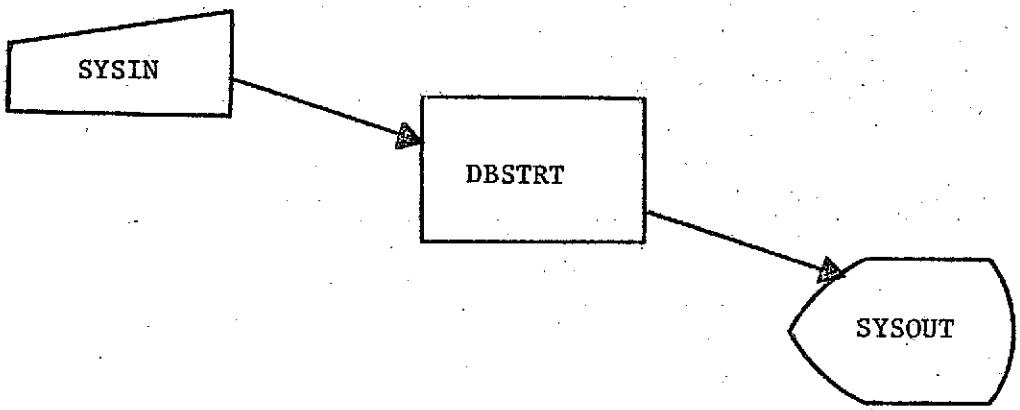


Figure 1. I/O Block Diagram

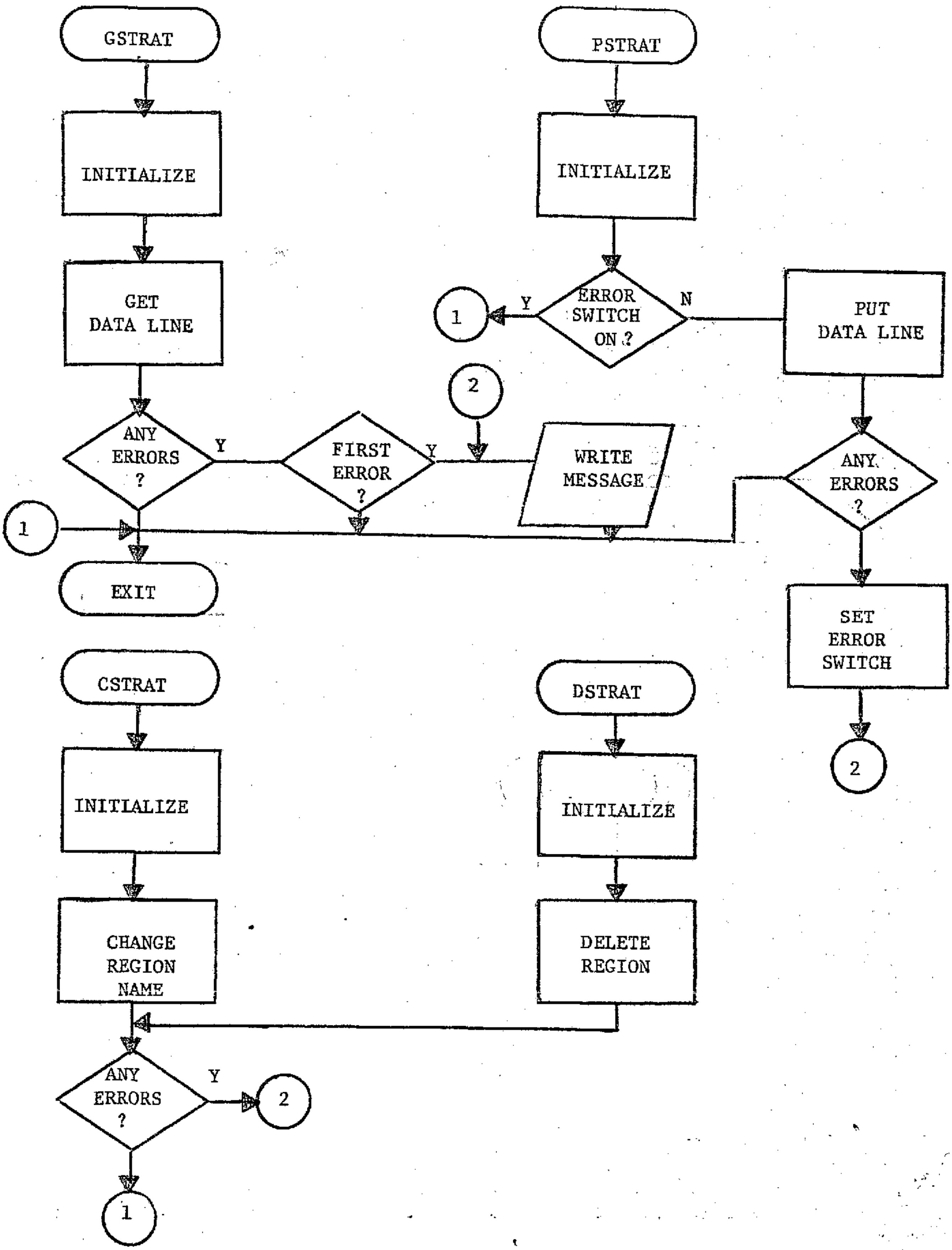


Figure 2A. Top Level Flowchart

TOPIC H.3 - STRATEGY ASSEMBLER ROUTINES

A. MODULE NAME

Program-ID - NTSTRAT
Module-ID - TSTRAT

B. ANALYST

Connie D. Becker
Neoterics, Inc.

C. MODULE FUNCTION

These routines act as the assembler service routines for the strategy library. They permit the retrieval, modification and storing of the saved strategies and formats.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Strategy data set - is used for input for both stored strategies and stored formats.

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      Strategy Data Set-is used for output for both stored strategies and stored formats.

   b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

USERTAB-is used to obtain the NASIS-ID.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. TSDELRG

At this entry point the program initializes itself to delete a strategy or format region. It opens the strategy data set, if necessary, and extracts the region name passed by the caller. The program then proceeds to delete the region, one line at a time. If any errors are encountered, the region name is set to null. The program then returns to the caller.

b. TSGETRG

At this entry point the program initializes itself to get a line from a strategy or format region. It extracts the parameter passed by the user, and if a null line number is passed, sets up to read the first line of the region. If the high order bit of the line number is off, it sets up to read the line following that indicated by the line number. Otherwise, it positions the file to the line number passed.

The program then attempts to read the line requested. If successful, it posts the line number, posts the data (with trailing blanks removed) and returns to the caller.

If an error occurs, the program sets the

region name to null before returning. Likewise, if an end-of-region occurs, the line number is set to null before returning.

c. TSPUTRG

At this entry point the program initializes itself to put a line to a strategy or format region. It opens the strategy data set, if necessary, and extracts the region, line number and data parameters passed by the caller. If the line number is null, it sets up to add the line at the end of the region. In any case, it positions the file to the proper region and live within the region. The program then attempts to write out the new line from the data passed by the caller. If successful the program simply returns to the caller. If an error occurs, the program sets the region name to null before returning.

d. TSCHGRG

At this entry point the program initializes itself to change the name of a strategy or format region. It opens the strategy data set if necessary, and extracts the old and new region names passed by the caller.

The program firsts attempts to delete any existing region with the new region name. If an error occurs, other than region unknown, the program terminates and sets the old region, reads a line, positions itself to the new region and writes out the live. This process is repeated until all of the data lines have been copied. If any errors occur, the new region is deleted, the old region name is set to null and the program returns to the caller. If no errors have occurred, the program deletes the old region and returns to the caller.

e. TSGETSN

At this entry point, the programs initializes itself to get a strategy region name. It opens the strategy data set, if necessary, and extracts the strategy name passed by the caller. If the name is null, the program sets up to get the first strategy name.

Otherwise, it sets up to get the strategy name following that passed by the caller.

The program then attempts to read a line from the strategy data set. If successful, it extracts the region name and passes that back to the caller. If an error occurs, or if an end-of-file is sensed, the region name is set to null and the program returns to the caller.

f. TSGETFN

At this entry point, the program initializes itself to get a format region name. It opens the strategy data set, if necessary, and extracts the region name passed by the caller. If the region name is null, the program sets up to get the first format name. Otherwise, it sets up to get the format name following that passed by the caller.

The program then attempts to read a line from both data sets. If an error occurs, or if both files indicate end-of-file, the region name is set to null and the program returns to the caller. Otherwise, the program compares the region names of the two lines. It posts the name, lowest in value, in the region name and returns to the caller.

F. CODING SPECIFICATION

1. Source Language

The module is written using the OS 360 Assembler language

2. Suggestions and techniques

Any output operation to the strategy data set results in the temporary closing of the data set, to ensure data set integrity in the event of a system crash.

TOPIC H.4 - USER VERB TABLE

A. MODULE NAME

Program-ID - NDBUSER
Module-ID - DBUSER

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This routine uses the currently defined verb table to locate any user defined commands for that table. If any have been defined, they are appended to the list already existing in the table.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

4. Reference Tables

VERBTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

Upon entry, the program tests for the presence of a VERBTAB. If none is found, it exits immediately. Otherwise, the program extracts the default symbol from the table and gets the default value for that symbol.

The program then begins analyzing the data, until none remains, at which time it returns to the caller. The data is expected in command-name and entry point pairs. Each pair is extracted from the data, analyzed for valid construction and then posted to the next available slot in the table.

If there are any syntax errors, invalid names, or if the table is filled, the program will return to the caller, bypassing the remaining entries.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the IBM PL/I language.

2. Suggestions and Techniques

Not Applicable

TOPIC H.5 - USER PROFILE ROUTINE

A. MODULE NAME

Program-ID - NDBPRO
Module-ID - DBPRO

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module performs the processing necessary for the implementation of the PROFILE, SYNONYM, DEFAULT, SYNONYMS and DEFAULTS commands.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-Line Terminal Entries

      The program prompts the user for the parameters required by the various commands.

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

The display of the user's defaults and synonyms produce formatted terminal displays.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

USERTAB-the program extracts the user's NASIS-id from the user data table.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBPRO

At this entry point the program analyzes the parameter value to determine the command to be processed; the appropriate label below is branched to for further processing.

b. DBPROF (PROFILE)

At this label the program simply calls TSPROF to write out a copy of the user's current profile. If any errors are detected, an appropriate diagnostic message is written to the user the program then terminates.

c. DBDEF (DEFAULT)

At this label the program initializes itself to process defaults. It repetitively prompts for data and calls TSPDEF to process the request. If any errors are encountered, an appropriate diagnostic message is written to the user. The program then terminates.

d. DBSYN (SYNONYM)

At this label the program initializes itself to process synonyms. It repetitively

prompts for data and calls TSPSYN to process the request. If any errors are detected, an appropriate diagnostic message is written to the user. The program then terminates.

e. DBDEFS (DEFAULTS)

At this label the program initializes itself to display the data values corresponding to a set of default symbols. The program also initializes the screen and paging control data. The program then attempts to read in the list of symbols. If no data was entered, the program sets up to display all of the default values. Otherwise it saves the list of symbols entered.

The program then repetitively calls TSGDEF for each entry in the list, to obtain its default value. The values are formatted and posted to the screen. When the screen is filled, or when the list of names is exhausted, the program displays the screen to the user, posts the paging status data and terminates.

f. DBSYNS (SYNONYMS)

At this label the program initializes itself to display the time values for a set of synonym terms. The program also initializes the screen and the paging control data. The program then attempts to read in the list of symbols.

If no data was entered, the program sets up to display all of the synonym values. Otherwise, it saves the list of symbols entered.

The program then repetitively calls TSGSYN for each entry in the list, to obtain its time value. The values are formatted and posted to the screen. When the screen is filled, or when the list of names is exhausted, the program displays the screen to the user, posts the paging status data and terminates.

g. DBPROPG (PAGE)

At this label the program re-initializes itself using the paging status data. If

data remains, the program branches to the proper routine to produce the next screen image. Otherwise, the program writes a diagnostic message and terminates.

F. CODING SPECIFICATIONS

1. Source Language

The program is written using the IBM 360 PL/I Language.

2. Suggestions and Techniques

Not Applicable

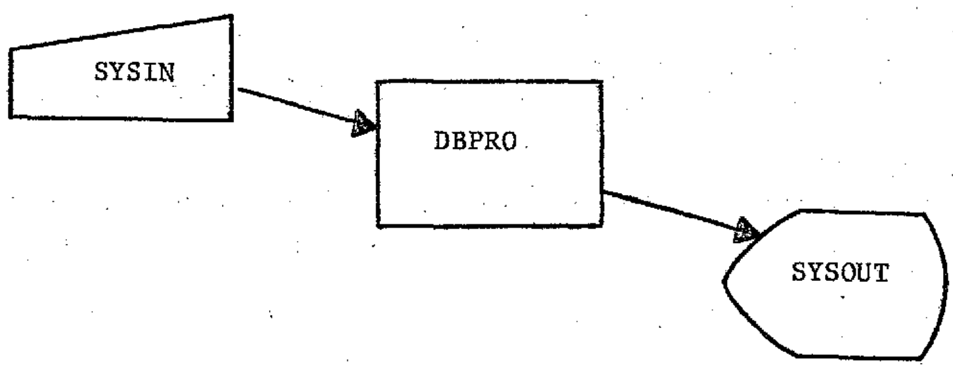


Figure 1. I/O Block Diagram

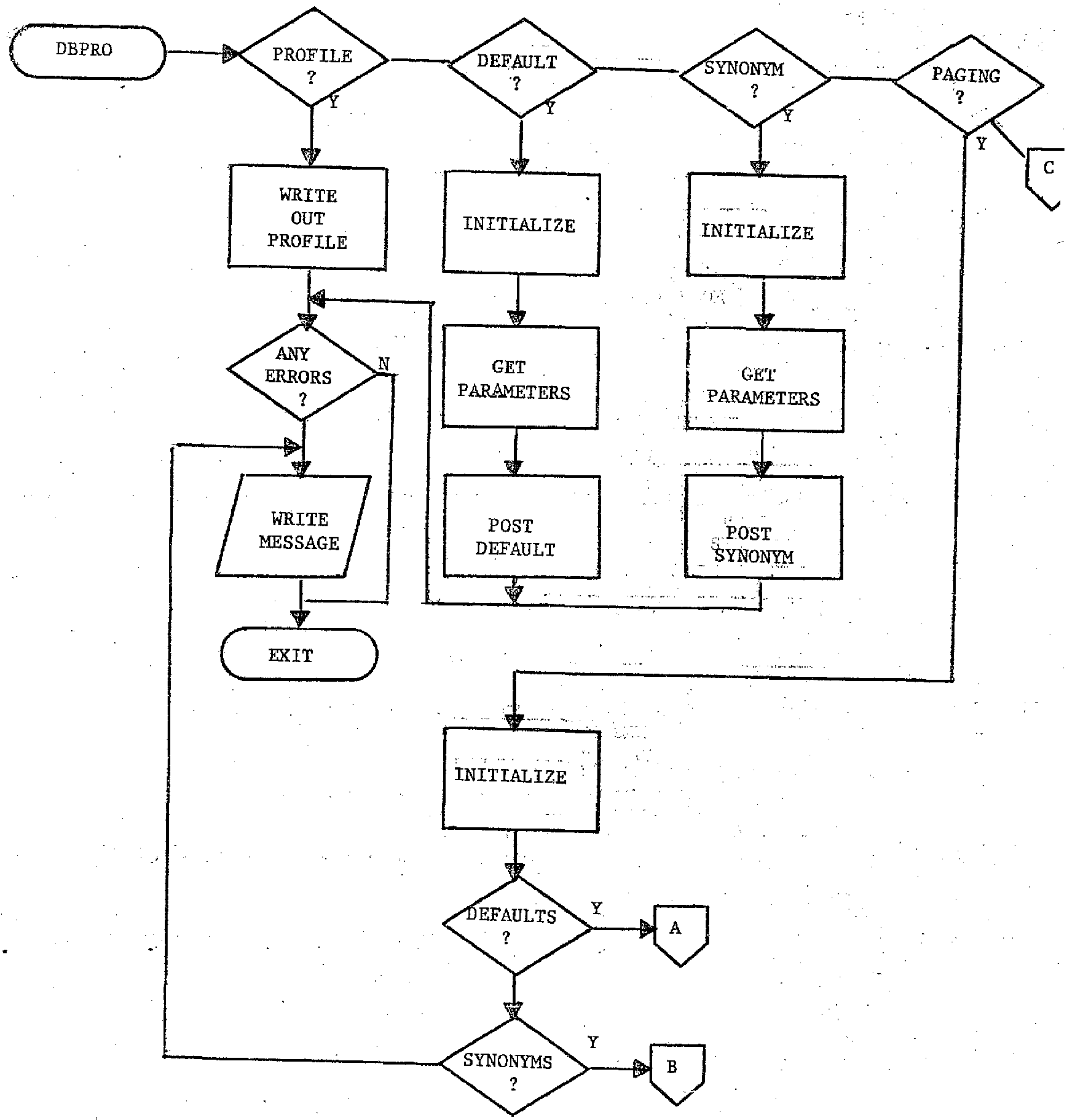


Figure 2Z. Top Level Flowchart - DBPRO

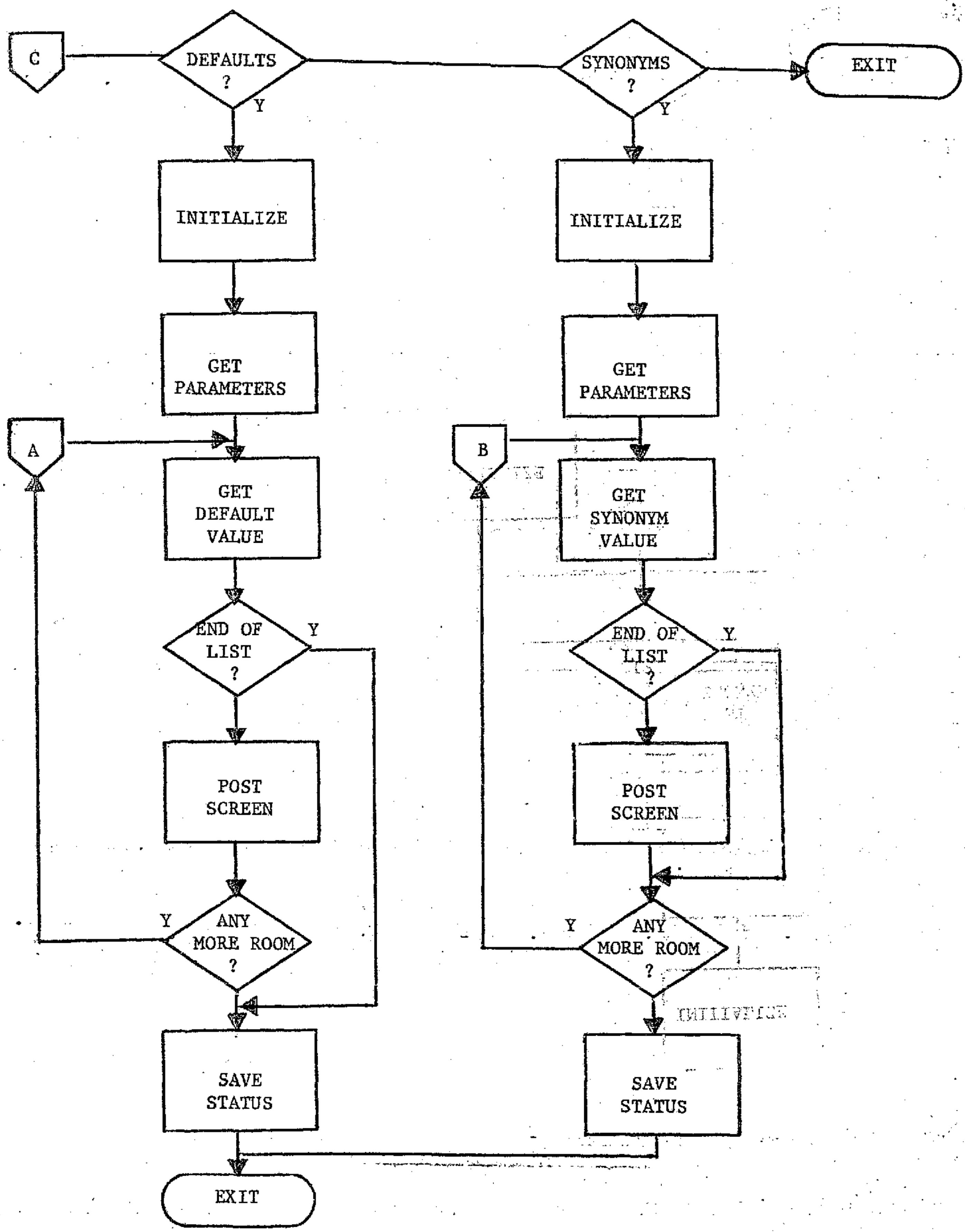


Figure 2B. Top Level Flowchart - DBPRO

TOPIC H.6 - USER PROFILE ASSEMBLER ROUTINES

A. MODULE NAME

Program-ID - NTSPRO
Module-ID - TSPRO

B. ANALYST

Connie D. Becker
Neoterics, Inc.

C. MODULE FUNCTION

These routines act as the assembler service routines for the user's profile. They permit the retrieval, modification and storing of all synonym and default values.

D. DATA REQUIREMENTS

1. I/O Block Diagram

   See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      PROFILE LIBRARY is used to initially obtain a profile for the user.

   d. On-line Terminal Entries

      Not Applicable

3. Output Data Sets

   a. Output Files

      PROFILE LIBRARY - the user's profile will be written out as a member of this library with the name of his NASIS-ID.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

e. Return Code

A return code will be posted with a value whose meaning is dependent upon the entry point called.

4. Reference Tables

USERTAB-the program extracts the user's NASIS-ID from the user data table.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. TSPROF

At this entry point the program initializes itself tc write out the current user's profile. It first allocates a new list and moves over all of the synonym entries not marked for deletion. It next moves over all of the default entries and re-orders the default data values. The program then attempts to locate an old profile for this user in the profile library. If one is found, it is deleted. The program then writes out the new profile and gives it the name of the user's NASIS-ID. If any errors are encountered the error code is posted. The program then returns to the caller.

b. TSGSYN

At this entry point the program initializes itself to retrieve a synonym value. It first searches the synonym entries until it locates

the logical location for the symbol specified. If the entry is present and has not been deleted, or if the entry located is the symbol whose abbreviation was specified, the synonym value is extracted and passed back to the caller. If the entry located did not correspond to the symbol specified, a null value is returned to the caller.

c. TSGDEF

At this entry point the program initializes itself to retrieve a default value. It first searches the default entries until it locates the logical location for the symbol specified. If the entry is present, the data value offset is located and the data value is moved into the caller's area. The program then returns to the caller.

d. TSPSYN

At this entry point the program initializes itself to post a synonym value. It first checks to see if this is a delete request. If not, the program builds the new entry. It then searches the synonym entries until it locates the logical location for the symbol specified. If the symbol is to be deleted and it is not present, the program returns immediately. Otherwise, it performs the deletion by copying the entries prior to the deleted entry and those following the deleted entry, to a new profile similarly. Similarly, adds are processed by inserting the added entry between the two list segments. Modifications, if allowed, are performed in place. If a new profile was created, the old list is deleted. If the request was not for a deletion, the program computes the minimum abbreviation length. If it was a deletion, all synonyms for the entry deleted are flagged as deleted. The program then returns to the caller.

e. TSPDEF

At this entry point the program initializes itself to post a default value. It first checks to see if this is a delete request. If not, the program builds the new entry. The program does a getmain for 1500 bytes. If the requested region is greater than 1500,

control is passed to ISPROF where the content profile is reorganized and entries flagged for deletion are deleted and the member is condensed. If the area now required is still greater than 1500, an error code of 10 is passed back to DBPRO, otherwise processing is continued. It then searches the default entries until it locates the logical location for the symbol specified. If the symbol is to be deleted and it is not present, the program returns immediately. Otherwise, it performs the deletion by copying the entries preceding the one to be deleted and those following it to a new profile. Similarly, adds are processed by inserting the added entry between the two list segments and appending the data value at the end of the profile. Modifications are performed in place, if possible, if not, the data value is simply added to the end of the profile. The program then returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the OS 360 Assembler language.

2. Suggestions and Techniques

The entry searching routine should be coded as a binary search and the list moving routine should be coded as efficiently as possible.

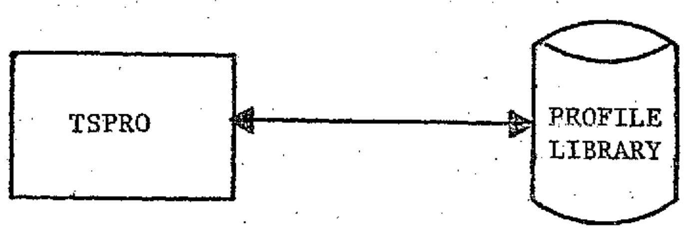


Figure 1. I/O Block Diagram

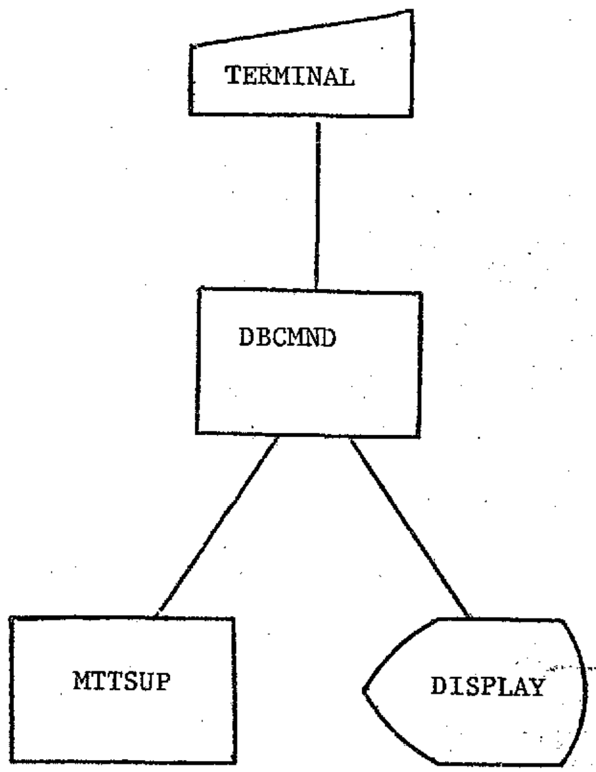


Figure 1. I/O Block Diagram

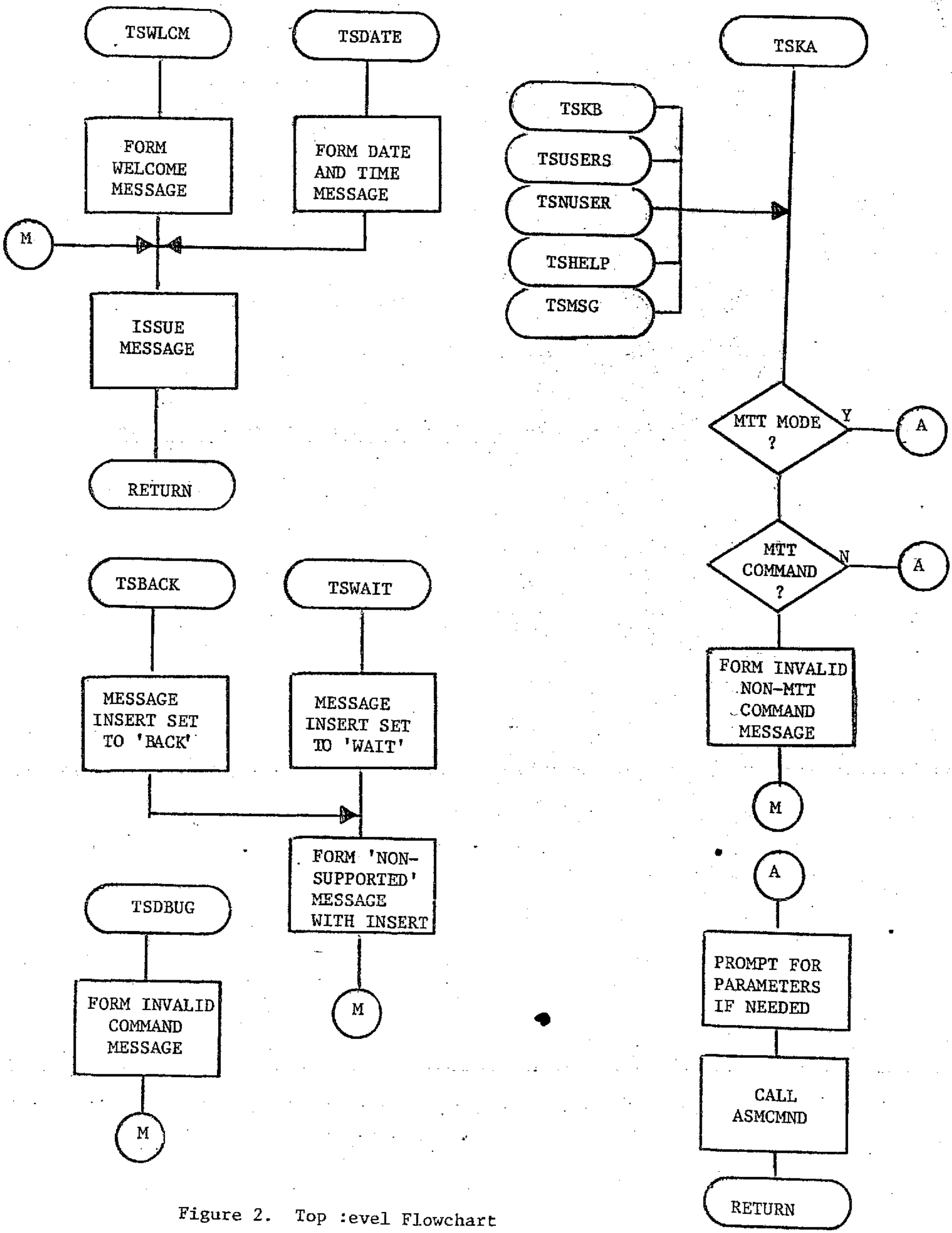


Figure 2. Top :evel Flowchart

TOPIC H.7 - IMMEDIATE COMMANDS, INTERFACE MODULE

A. MODULE NAME:

Program-ID: NDBCMND
Module-ID: DBCMND
Entry Points: TSWLCM, TSBACK, TSDBUG, TSKA, TSKB,
TSUSERS, TSNUSER, TSDATE, TSWAIT,
TSHELP, TSMSG

B. ANALYST

William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

This module contains the PL/I entry points for ten of the immediate commands. In addition, the TSWLCM entry writes the 'NASIS WELCOMES YOU' message to the user terminal at LOGON. The immediate commands KA, KB, USERS, NUSERS, HELP, and MESSAGE are set up in this module and then processed by the monitor (MTTSUP) via a call to ASMCMND. The immediate commands BACK, $DEBUG, DATETIME, and WAIT are processed by this module only.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

The program prompts for the parameters of the various applicable immediate commands.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program issues various informative and diagnostic messages to the user's terminal.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program references the USERTAB table.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2.

2. Narrative

a. Monitor-supported immediate commands

The immediate commands serviced by the entry points TSKA, TSKB, TSUSERS, TSNUSER, TSHELP, TSMSG are processed by the monitor via a call to ASMCMND. However, the mode of the current user, MTT or non-MTT, is checked so that if a non-MTT user tries to use a MTT command he is given a diagnostic. A return is made to the calling program.

b. Unsupported immediate commands

The entry points TSBACK, TSWAIT, and TSDBUG are diagnosed as unsupported commands.

c. TSWLCM

This entry point is called by DBMTT when a user logs on. This entry point provides the user with the welcome message and the date.

d. TSDATE

This entry point performs the DATETIME function within this module. The DATE function is used from PL/I and Zeller's congruence is used to determine the day of the week. The message is formed and sent to the user's terminal. A return is then made to the calling program.

F. CODING SPECIFICATIONS

1. Source Language

This program is written in PL/I (F) using the TSPL/I preprocessor.

2. Suggestions and Techniques

Not Applicable